For Macintosh Programmers & Developers

MacTech™ MAGAZINE

INSIDE: Macintosh Programming FAQs

Vol. 13, No. 2 • February 1997

Games Galore!



APPLE'S NEW GAME SPROCKETS

PLUS Programmer's Challenge: Othello™

$5.85 US
$6.95 Canada
ISSN 1067-8360
Printed in U.S.A.

02
0 32128 74887 8

Eddy Award Winner** for **Best New Developer Tool
– MacUser Editors Choice Awards, 1993

"A distinct improvement over ResEdit."
– MacTech / MacTutor

"Resorcerer's data template system is amazing!"
– Bill Goodman, author of Compact Pro

"Nuke ResEdit! Resorcerer is mission-critical for us."
– Dave Winer, Userland Frontier

"The color pixel editors are wonderful! A work of art!"
– Dave Winzler, author of Microseeds Redux

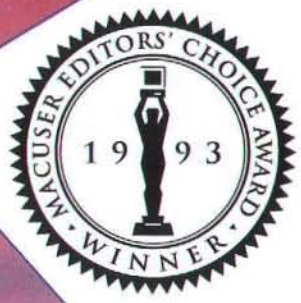
"Every Macintosh developer should own a copy of Resorcerer."
– Leonard Rosenthol, Aladdin Systems

"Resorcerer will pay for itself many times over in saved time and effort."
– MacUser review

"The template that disassembles 'PICT's is awesome!"
– Bill Steinberg, author of Pyro! and PBTools

"Resorcerer proved indispensible in its own creation!"
– Doug McKenna, author of Resorcerer

"...a wealth of time-saving tools."
MacUser Review, Dec. 1992



RESORCERER®

Version 1.2.4

The Resource Editor for the Macintosh Wizard

ORDERING INFO

Needs: ≥Mac Plus, ≥ Sys 4.2, 1MB
Likes: ≥Mac Plus, ≥ Sys 7.0, 2MB
32-bit clean, AU/X compatible

Price: $256 (decimal)
(Educational, quantity, or other discounts available)

Includes: 500 page manual
60-day Money-Back Guarantee
Domestic UPS ground shipping

Payment: Check, PO's, or Visa/MC

Extras (call us):
COD, FedEx, UPS Blue/Red, International Shipping

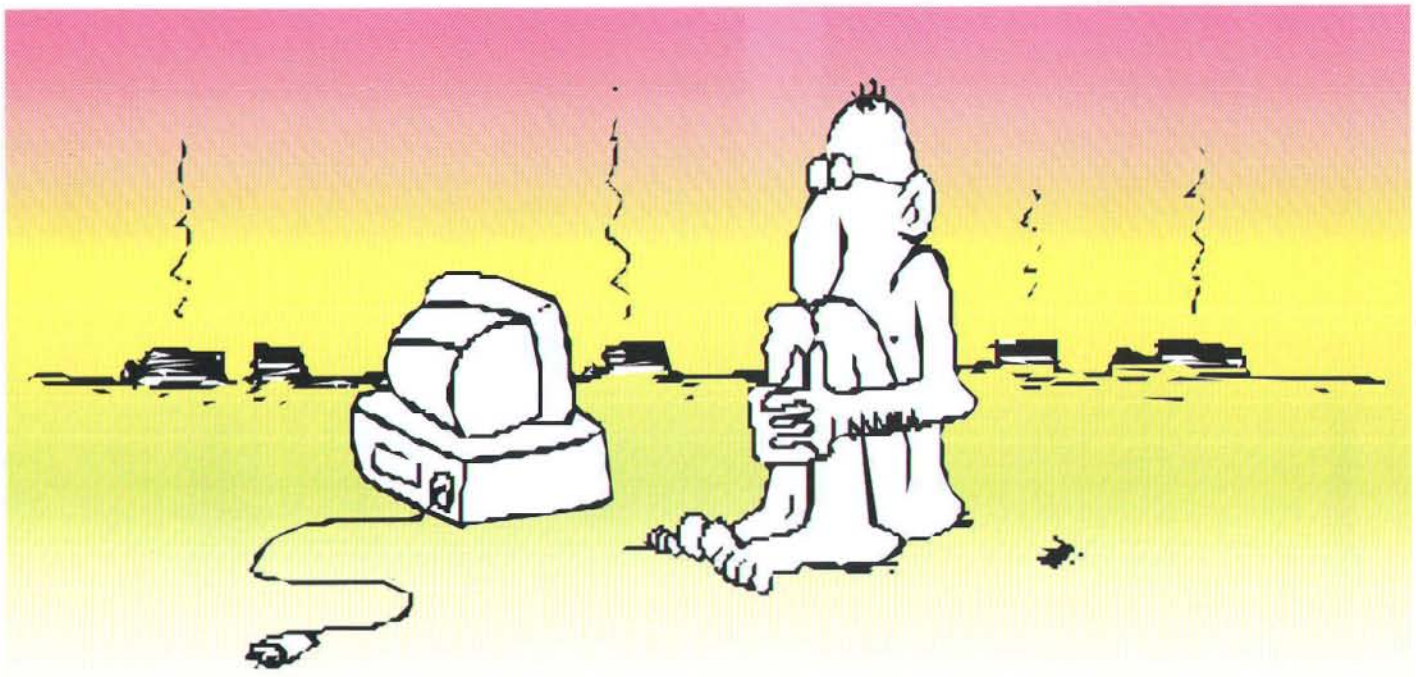
Downloadable Demos/Updaters:
AppleLink: Software Sampler
AOL: Software Libs/Development
CompuServe: MACDEV/Tools
or call us.

New 1.2 Features:

- **New 'cicn', 'ppat', 'crsr', 'acur', 'pltt', 'clut' editors**
- **Powerful icon family editing (all 9 icon types)**
- **Color pixel anti-aliasing, dithering, and lots more**
- **Complete 'PICT' disassembly and reassembly**
- **Resource sorting; ROM resource browsing**
- **120 template field parsing types now supported**
- **New insertion & deletion template field types**
- **Text-only 'PICT' resources**
- **Lots of improvements throughout**

- Easier, faster, more Mac-like, and more productive than ResEdit
- Safer memory-based, not disk-file-based, design and operation
- All file information and common commands in one easy-to-use window
- Compares resource files, and even **edits your data forks** as well
- Visible, accumulating, editable scrap
- Searches and opens/marks/selects resources by text content
- Makes global resource ID or type changes easily and safely
- Builds resource files from simple Rez-like scripts
- Most editors DeRez directly to the clipboard
- All graphic editors support screen-copying or partial screen-copying
- Hot-linking Value Converter for editing 32 bits in a dozen formats
- Its own 32-bit List Mgr can open and edit very large data structures
- Templates can pre- and post-process any arbitrary data structure
- Includes nearly 200 templates for common system resources
- TMPLs for Installer, MacApp, QT, Help, AppleEvent, OCE, GX, etc.
- Full integrated support for editing color dialogs and menus
- Try out balloons, 'ictb's, lists and popups, even create C source code
- Integrated single-window Hex/Code Editor, with patching, searching
- Editors for cursors, versions, pictures, bundles, and lots more
- Well-designed, helpful developer tools being added all the time
- Relied on by thousands of Macintosh developers around the world

MATHEMÆSTHETICS, INC.
P.O. Box 298 • Boulder • CO • 80306-0298 • USA
Phone: (303) 440-0707 • Fax: (303) 440-0504
AppleLink/AmericaOnline: RESORCERER • Internet: resorcerer@aol.com



Do I have to use a PC to port Windows applications to a Mac?

WILLOWS TWIN APIW for Macintosh

Windows API tool box extension for Macintosh developers

- Multi-platform implementation of the Windows API
- Works with Macintosh tools and compilers
- Native library optimized for PowerPC or 68K
- Work within the IDE of your choice

Call now: (408) 777-1820
Visit our web site: http://www.willows.com

WILLOWS SOFTWARE

Willows Software, Inc. 12950 Saratoga Avenue, Suite A, Saratoga, CA 95070

All trademarks are trademarks of their respective companies.

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

A publication of XPLAIN CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 800-MACDEV-1

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@devdepot.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

MACTECH MAGAZINE

MacTech Magazine is grateful to the following individuals who contribute on a regular basis. We encourage others to share the technology.

We are dedicated to the distribution of useful programming information without regard to Apple's developer status.

For information on submitting articles, ask us for our ***writer's kit*** *which includes the terms and conditions upon which we publish articles.*

Editorial Board of Advisors

Scott T Boyd, Jordan J. Mattson, Jim Straus, and Jon Wiederspan

Editors

Publisher • Neil Ticktin
Editor-in-Chief • Eric Gundrum
Managing Editor • Jessica Courtney
Online Support • Nick "nick.c" DeMello

Contributing and Technical Editors

Java • Will Iverson, Apple Computer, Inc.
Internet • Carl de Cordova, Apple Computer, Inc.
Mac OS 8 • Steve Kiene, Mindvision
MacDev-1™ • Rich Siegel, Bare Bones Software, Inc.
MagicCap/TeleScript • Richard Clark, General Magic
OpenDoc • Tantek Çelik, 6prime Corporation
Performance Programming • Jim Gochee, Connectix
Product Reviews • Ed Ringel

Regular Columnists

Getting Started • Dave Mark
Programmer's Challenge • Bob Boonstra
Symantec Top 10 • Symantec Technical Support
From the Factory Floor • Dave Mark, Metrowerks
Tips & Tidbits • Steve Sisak
URLs • Nick "nick.c" DeMello

XPLAIN CORPORATION

Chief Executive Officer • Neil Ticktin
Chief Operating Officer • Andrea J. Sniderman
Advertising Executive • Ruth Subrin
Developer Depot Manager • Joshua Miller
Marketing Manager • Susan M. Worley
Customer Relations • Susan Pomrantz, Erik Davidson
Accounting • Annette Perez, Jan Webber
Network Administrator • Donal Corcoran

Financial Services • Shin Financial Services
Art Direction/Production • InfoGraphix

Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

PRINTED WITH SOY INK

This publication is printed on paper with recycled content.

All contents are Copyright © 1984-1997 by Xplain Corporation. All rights reserved. MacTech, MacTech Magazine, MacTech CD-ROM, MacTech Web, THINK Reference, Developer Depot, Sprocket, JavaTech, WebTech, MacDev-1, MacTech NOW, Developer Central, Virtual Developer Central and the MacTutorMan are trademarks of Xplain Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

TABLE OF CONTENTS

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

February 1997 • Volume 13, No.2

FEATURE ARTICLES

GAME DEVELOPMENT

Sprockets are Forever

Apple's new Game Sprockets provides a simple, reliable interface to the great game features of the MacOS

— By Jeremy Vineyard 12

Shift into High Gear with Apple Game Sprockets

Turning your game idea into a reality, step by step

— By Jamie Osborne 30

COMMUNICATIONS TOOLBOX

HappyFace

Using the Communications Toolbox to build an interactive, two-player game

— By Mark Chally 44

TOOLBOX TECHNIQUES

The Language of Resources

A common sense introduction to the Resource Manager

— By Chris Stasny 54

PROGRAMMING TECHNIQUES

Serial Killer

Easy and effective serial number management for electronic game distributors

— By Jamie McCornack 58

TOOLS OF THE TRADE

Macintosh Programming FAQs

— By Edward Ringel 74

REGULAR COLUMNS

GETTING STARTED

The ShapeWorld Applet — *By Dave Mark* 6

PROGRAMMER'S CHALLENGE

Othello™ — *By Bob Boonstra* 16

FROM THE FACTORY FLOOR

Mike Lockwood, Debugger — *By Dave Mark* 62

SYMANTEC TOP TEN

— By Kevin Kenan 65

UNIFORM RESOURCE LOCATORS

— By Nick "nick.c" DeMello 69

VIEWPOINT 4

THE CLASSIFIEDS 49

NEWSBITS 70

TIPS & TIDBITS 73

ADVERTISER & PRODUCT INDEX 78

VIEWPOINT



By Eric Gundrum

The Mac is a Game Machine

Back when the Mac was just starting out it was heavily derided as just a toy, a game machine, by the business folks who didn't want to see the Mac encroach on their control of computing power. These are the people who thought using a computer had to be hard work if the computer was capable of working hard. The Mac, with its smiling acceptance of a floppy disk, its high quality graphics, its ease of use, just couldn't be anything but a game machine; it was too much fun!

Apple fought long and hard against that stigma. They ostracized games developers, driving all but the most determined of them to other platforms. It took Apple several years, but they were successful. Macs had made it into the big time, with full corporate acceptance, and very few games. The Macintosh community had invented desktop publishing.

With so many businesses soon using Macs, people began to take notice of just how much fun it was to use this little computer. After all, what other computing platform had flying toasters, and with Opus to shoot them down? Desktop entertainment was born, and its mother was Macintosh. Still, Apple insisted that the Mac was not a game machine.

Several more years passed and the Macintosh community grew larger. Many new flavors of Macintosh were now available, each faster than its predecessor, with fancy color screens, CD-ROM drives and video in and out. Still, it wasn't a game machine. Sure there were a few games available, often tucked away as screen saver modules or poorly implemented ports from another platform. Yet none of these programs really took advantage of the unique features of the Macintosh, and features like multi-channel stereo sound had been with the Mac for several years.

Finally, with the advent of the PowerMac, Apple started to accept that people wanted to play games, and they wanted to do so with their Mac. A few games appeared to explore the power of CD-ROM, stereo sound, superb graphics and PowerPC. However, the Macintosh was still not the platform of choice for game developers.

Yet more time passed. The Mac has been pushed aside by some of the more short-sighted business folks, but Apple — yes, Apple — has finally said that the Mac *is* a game machine.

Game Sprockets, a Great Idea

With the advent of Apple's Game Sprockets, Apple provided game developers a set of tools making programming quality games for Macintosh easy, providing access to the Mac's powerful multimedia capabilities while helping the games become more consistent and reliable.

As you will undoubtedly see from reading the articles in this issue, Apple's Game Sprockets are a powerful tool. They provide simple and reliable access to double buffered animation, multi-channel stereo sound, protocol independent network gaming, and support for an infinite variety of input devices. Game Sprockets also handle much of the mundane user interface to set Macintosh hardware options, including choosing an input device, display device and network players. Add in the many incredible multimedia technologies already available on the Mac, and the Macintosh platform becomes one heck of a cool game machine.

But, Why Support Games?

Some people complain that Apple shouldn't waste resources on foolish things like helping game developers. The reality is that people love playing games. People are often so fascinated by computer games that they will play them repeatedly for several hours and not realize where their time has gone.

Games generally demonstrate more creativity than any other software genre. Often they push the limits of the computer's capabilities beyond where anyone thought they could go. It is more likely that games are responsible for the existence of 10x speed CD-ROM drive than is looking up phone numbers. The Mac has had very few new input devices, but Apple's Input Sprocket makes the porting of innovative input hardware from other platforms much easier, and increases the size of the potential market.

Look carefully at the technology of Game Sprockets. Is it really for use only in games? The input devices could be used in a variety of interactive multimedia applications. There is nothing about the platform independent, client-server network capabilities that checks for alien invader information in the exchanged messages. Double-buffered video has been used for years to accelerate many graphics applications; now it is more reliable. Game Sprockets may make these and other capabilities much easier even for non-entertainment applications.

In This Issue

In this issue we offer two new Macintosh games, complete with source code. Try 'em; if you don't like them, well, you have the sources, fix them to work the way you think they should work.

This issue also includes an article suggesting how to implement serialization of your application, making network-based product distribution more profitable. For those of you still learning the basics, we include an article describing how to use the Resource Manager. If you are keen on communications technologies, check out the game article built around the use of the Communications Toolbox.

With all this cool technology, the excuses for so few Mac games are dwindling. I look forward to many new cool apps to choose from by next holiday shopping season. MT



MORE DEVELOPERS PROTECT.





MacHASP® PROTECTS MORE.

Mac™OS

MacHASP Packs More Into Less.

Based on state-of-the-art ASIC technology, MacHASP packs the industry's most advanced security, compatibility and flexibility into a compact and end-user-friendly dongle.

Grow With Aladdin!

The fastest growing company in the industry, with over 4 million keys sold to 20 thousand developers worldwide, Aladdin is setting the standard for software security today.

NSTL Study Rates HASP No.1!

A recent test conducted by the National Software Testing Labs*, the world's foremost independent lab, compared the flagship PC products of leading software protection vendors. The result? HASP was rated the clear overall winner - and number one in all the major comparison categories. For a full copy of the NSTL report, contact your local HASP distributor.

These days, more and more developers are choosing to protect their software against piracy. They're protecting more products, on more platforms, with better protection – and selling more as a result.

And more of these developers are protecting with MacHASP. Why? Because MacHASP offers more security, more reliability and more features than any other product on the market. Only MacHASP offers capabilities such as network support, anti-debugging envelope protection, and secure remote activation and updating.

MacHASP supports the most advanced platforms, including all versions of MacOS and Power Mac – as well as AppleTalk networks. And because Aladdin is a licensed Apple Partner, MacHASP guarantees full, transparent compatibility with the ADB standard.

To learn more about how you can protect better – and sell more – call now to order your MacHASP Developer's Kit.



The MacHASP Developer's Kit contains everything you need to protect your software today!

1-800-223-4277

w w w . a k s . c o m

ALADDIN™

The Professional's Choice

North America	**Aladdin Knowledge Systems Inc.**	Tel: (800) 223 4277, 212-564 5678, Fax: 212-564 3377, E-mail: hasp.sales@us.aks.com
Int'l Office	**Aladdin Knowledge Systems Ltd.**	Tel: +972-3-636 2222, Fax: +972-3-537 5796, E-mail: hasp.sales@aks.com
Germany	**FAST Software Security AG**	Tel: +49 89 89 42 21 37, Fax: +49 89 89 42 21 40, E mail: info@fast-ag.de
United Kingdom	**Aladdin Knowledge Systems UK Ltd.**	Tel: +44 1753 622266, Fax: +44 1753 622262, E mail: sales@aldn.co.uk
Japan	**Aladdin Japan Co., Ltd.**	Tel: +81 426-60 7191, Fax: +81 426-60 7194, E-mail: sales@aladdin.co.jp
Benelux	**Aladdin Software Security Benelux B.V.**	Tel: +31 24-641 9777, Fax: +31 24-645 1981, E-mail: 100526.1356@compuserve.com

■ **Aladdin Russia** 095 9230588 ■ **Australia** Conlab 03 98985685 ■ **Chile** Micrologica 02 7350041 ■ **China** Shanghai UR 021 6437 7828 ■ **Czech** Atlas 02 766805 ■ **Denmark** Berendsen 039 577316 ■ **Egypt** Zemelden 02 3604632 ■ **Finland** ID-Systems 0 8703520 ■ **France** 1 40859885 ■ **Greece** Unibrain 01 6756320 ■ **Hong Kong** Hastings 02 5484629 ■ **India** Solution 011 2148254 ■ **Italy** Partner Data 02 26147380 ■ **Korea** Dae A 02 8484481 ■ **Mexico** SiSoft 91800 55283 ■ **New Zealand** Training 04 5666014 ■ **Poland** Systherm 061 480273 ■ **Portugal** Futurmatica 01 4116269 ■ **Romania** Ro Interactive 064 140283 ■ **Singapore** ITR 065 5666788 ■ **South Africa** D.t.e Roux 011 8864704 ■ **Spain** PC Hardware 03 4493193 ■ **Switzerland** Opag 061 7169222 ■ **Taiwan** Teco 02 5559676 ■ **Turkey** Mikrabota 0312 4670635 ■ **Yugoslavia** Asys 021 623920

© Aladdin Knowledge Systems Ltd. 1985-1996. (10.96) MacHASP® is a registered trademark of Aladdin Knowledge Systems Ltd. All other product names are trademarks of their respective owners. Mac & the Mac OS logo are trademarks of Apple Computer, Inc., used under license. NSTL makes no recommendation or endorsement of any product. †The NSTL report was commissioned by Aladdin.



By Dave Mark

The ShapeWorld Applet

A few months ago I promised an applet that did double-buffered animation. When I started work on the sample applet, I realized that I was definitely jumping the gun and there was still a lot of ground to cover to get to double-buffering. Since then, I've explored the AWT event-handling mechanism, as well as the various shape drawing routines in the Graphics class.

This month I am going to bring these concepts (and a few new ones) together into an applet called ShapeWorld. This month's version of ShapeWorld (I'll extend it next month) randomly scatters some rectangles into a Canvas, then responds to mouse clicks by selecting and deselecting the rectangles. As you'll see next month, ShapeWorld was designed to be extended. For now, let's get the first version up and running.

The ShapeWorld Project

Launch the CodeWarrior IDE and create a new project named ShapeWorld.μ using the Java applet stationery. Create a new source file named ShapeWorld.java and add it to the project. Here's the ShapeWorld.java source code:

```
import java.awt.*;
import java.util.*;

abstract class Shape
{
  boolean     highlighted;
  ShapeCanvas shapeCanvas;
  int         shapeX, shapeY, shapeWidth,
shapeHeight;
  Rectangle   boundsRect;
  Shape( ShapeCanvas canv, int x, int y,
                int width, int height )
  {
    shapeCanvas = canv;
    highlighted = false;

    shapeX = x;
    shapeY = y;
    shapeWidth = width;
    shapeHeight = height;

    boundsRect = new Rectangle( x, y, width, height );
  }

  abstract public void draw( Graphics g );

  public void setHighlight( boolean newHighlight )
  {
    highlighted = newHighlight;
  }

  public boolean isHighlighted()
  {
    return highlighted;
  }

  public boolean isPointInShape( int x, int y )
  {
    return boundsRect.inside( x, y );
  }
}

class RectShape extends Shape
{
  RectShape( ShapeCanvas canv, int x, int y,
                int width, int height )
  {
    super( canv, x, y, width, height );
  }

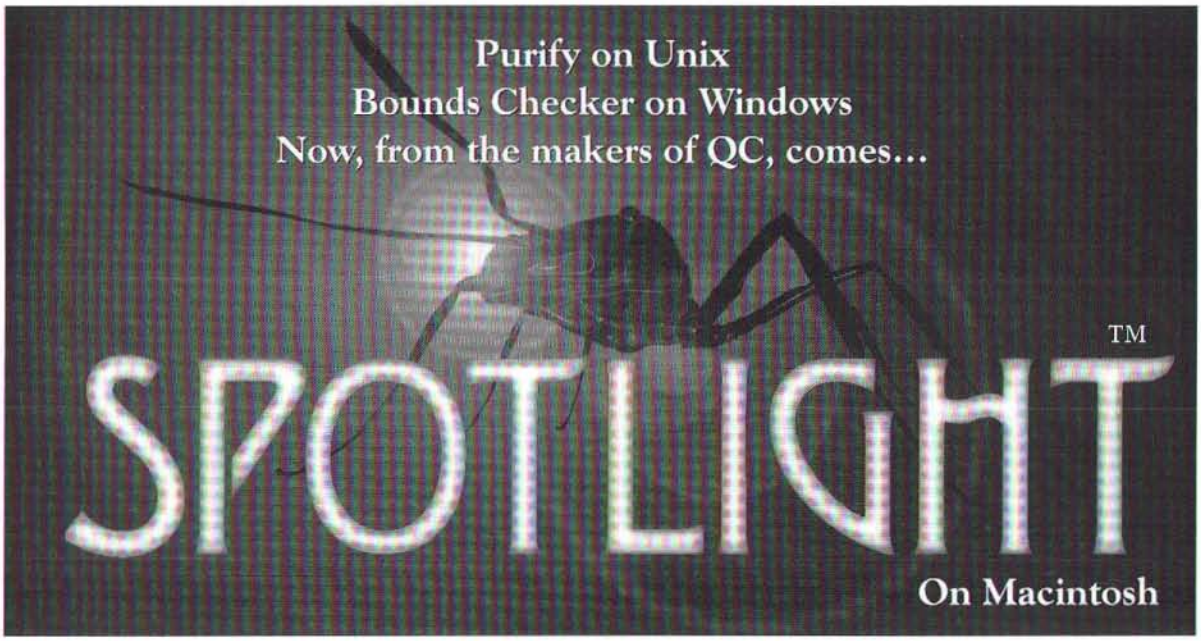
  public void draw( Graphics g )
  {
    if ( isHighlighted() )
    {
      g.setColor( Color.black );
      g.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
      g.setColor( Color.red );
      g.fillRect( shapeX+2, shapeY+2,
                  shapeWidth-4, shapeHeight-4 );
    }
    else
    {
      g.setColor( Color.red );
      g.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
    }
  }
}

class ShapeCanvas extends Canvas
{
  Vector   shapes;
  Shape    curShape;

  ShapeCanvas( int width, int height )
  {
    shapes = new Vector();
    curShape = null;

    setBackground( Color.yellow );

    resize( width, height );
  }
```



Find Bugs Fast

Spotlight is the first **Automatic Memory Debugger** for the Macintosh. Instantly detect invalid memory accesses, bad toolbox parameters, leaks, stack overwrites, memory relocation problems, and much more.

Spotlight uses your XSYM file to automatically patch your application — no need to change your source code. No need to recompile. **No learning curve whatsoever.**

The interface gives you instant feedback when an error is detected. You can ignore the error, ignore all future occurrences of the error, or log the error to a text file for later analysis.

Fine Grained Memory Protection

Spotlight identifies reads and writes outside of the application and system heap. But it does not stop there. Spotlight's Object Code Replacement instrumentation **inspects each read and write instruction in your code,** detecting faulty memory reads and writes that occur between blocks, in released blocks, across multiple blocks, and in ROM.

This technology works on any heap object you can allocate: Mac heap objects, C 'malloc' heap objects, even C++ objects created with new.

Spotlight stops your application whenever a faulty memory access is about to occur and displays the **exact source code line** where it will happen. The calling stack is shown along with a display of all variables. A memory viewer shows the erroneous memory address and contents.

Toolbox Validation

Spotlight checks parameters to **over 400 toolbox API calls,** automatically validating handles, memory blocks, return values, and so on. Specific checks catch subtle errors such as drawing into an unlocked GWorld, passing an invalid window pointer, passing an address within an unlocked handle to a routine that may move memory, and too many more to list.

Leak Detection

No more struggling with MacsBug. On program exit Spotlight provides a **clear listing of all leaked memory** showing a full stack trace from where the memory was allocated. Instantly discover all leaked Mac OS objects, malloc'd objects, C++ objects, and resources.

Limitations

Spotlight requires an XSYM file to function. MPW and CodeWarrior users can generate these directly. Symantec users must use ToolServer to link with an XSYM capable linker. Spotlight requires a PowerPC processor.

Spotlight works on the object code. **No source code is required**: just the thing for testing purchased third party libraries.

Availability and Pricing

Pricing for Spotlight DR1 is $199 US (plus $5 shipping and handling within the continental US, $15 for international orders). This includes a free upgrade to the GM version and access to ftp interim upgrades leading up to GM. QC users can cross-grade to Spotlight for only $149.

All Onyx products carry a 30 day no questions asked money back guarantee.



Onyx Technology, Inc.
7811 27th Avenue West
Bradenton, Florida 34209

sales@onyx-tech.com www.onyx-tech.com
941 795-7801 941 795-5901 (fax)

Spotlight and QC are trademarks of Onyx Technology, Inc. All other trademarks are property of their respective owners.

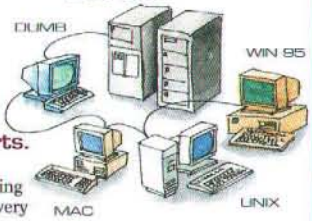
PLATFORMS: APPLE SYSTEM 7 • APPLE A/UX •
WINDOWS NT • WINDOWS 95 • OS/2 • NLM • DOS • LYNX • SCO • SYSTEM MANAGER • SUN SPARC-SOLARIS • IBM RS/6000
NT (ALPHA) • MIPS ABI (SGI) • AT&T UNIX • BANYAN • INTERACTIVE UNIX • LINUX • MOTOROLA 88OPEN • QNX •
• HP9000 • RS/6000 • SUN O/S 4.X •

Q: What does it take to deploy a superior client/server application?
A: *A SUPERIOR SERVER*

START with the most advanced client-side SDK on the market: c-tree® Plus at $895.
- **Complete "C" Source code**
- ROYALTY FREE (Client Side)
- Multiple supported protocols
- Fast, portable, reliable
- Powerful features like transaction processing
- Win95, NT, and Windows 3.1 ready

RESULT?
A solid, economical, easily deployable product that fits your needs.
- Portable
- Scalable
- Exceptional Performance
- Flexible
- Easy Server distribution
- Convenient OEM terms

ADD a strong, multi-platform, industrial-strength Server that supports.
- File mirroring
- Heterogeneous networking
- Automatic disaster recovery
- Multi-threaded design
- Best price/performance available: from $445- $3745



Heterogeneous TCP/IP Network

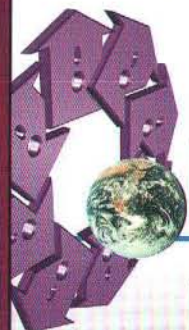
You can't find a better client SDK with these features!
Over sixteen years of proven reliability and performance.
No one else supports over 30 platforms in this price range!

c-tree Plus®
- Complete C Source
- Single/Multi User
- Client/Server (optional)
- Full ISAM functionality
- No Royalties
- Transaction Processing
- Fixed/Variable Length Records
- High Speed Data/Index Caching
- Batch Operations
- File Mirroring
- Multiple Contexts
- Unsurpassed Portability

FairCom® Server
- Client/Server Model
- Transaction Processing
- Requires <2MB RAM
- Online Backup
- Disaster Recovery
- Rollback - Forward
- Anti-Deadlock Resolution
- Client-side "C" Source
- Multi-threading
- Heterogeneous networking
- File Mirroring
- OEM/Source Available

FOR YOUR NEXT PROJECT CALL FAIRCOM: YOU CAN'T FIND A BETTER HETEROGENEOUS CLIENT/SERVER SOLUTION!

Also inquire about these FairCom products:
d-tree™
r-tree®
ODBC Driver



FAIRCOM® CORPORATION
Since 1979
WWWeb Address: http://www.faircom.com/
800-234-8180
U.S.A. phone (573) 445-6833 fax (573) 445-9698
EUROPE phone (035) 773-464 fax (035) 773-806
JAPAN phone (0592) 29-7504 fax (0592) 24-9723

```
    public void addShape( Shape newShape )
    {
      shapes.addElement( newShape );
    }

    public void paint( Graphics g )
    {
      for ( Enumeration e = shapes.elements();
                  e.hasMoreElements(); )
      {
        Shape s = (Shape)e.nextElement();
        s.draw( g );
      }
    }

    public Shape findInShapeList( int x, int y )
    {
      for ( Enumeration e = shapes.elements();
                  e.hasMoreElements(); )
      {
        Shape s = (Shape)e.nextElement();

        if ( s.isPointInShape( x, y ) )
        {
          s.setHighlight( ! s.isHighlighted() );
          s.draw( getGraphics() );

          return s;
        }
      }

      return null;
    }

    public void update (Graphics g)
    {
      paint(g);
    }

    public boolean mouseDown( Event e, int x, int y )
    {
      curShape = findInShapeList( x, y );

      return true;
    }
}

public class ShapeWorld extends java.applet.Applet
{
    ShapeCanvassCanvas;
    final int  shapeWidth = 20;
    final int  shapeHeight = 20;

    public void init()
    {
      int      x, y;

      sCanvas = new ShapeCanvas( 440, 290 );
      add( sCanvas );

      Random ran = new Random();
      Rectangle b = sCanvas.bounds();

      for ( int i=1; i<=10; i++ )
      {
        x = b.x + (int)((float)(b.width) * ran.nextFloat() );
        if ( x > b.x + b.width - shapeWidth )
          x -= shapeWidth;

        y = b.y + (int)((float)(b.height) * ran.nextFloat() );
        if ( y > b.y + b.height - shapeHeight )
          y -= shapeHeight;

        RectShape r = new RectShape( sCanvas, x, y,
                  shapeWidth, shapeHeight );
        sCanvas.addShape( r );
      }
    }
}
```

Save your typing, then create a new source code file named ShapeWorld.html and add *it* to the project. Here's the html:

```
<title>ShapeWorld</title>
<hr>
<applet codebase="ShapeWorld Classes" code="ShapeWorld.class"
width=500 height=300>
</applet>
<hr>
<a href="ShapeWorld.java">The source.</a>
```

Running the ShapeWorld Applet

Once your source and html are entered and saved, run the ShapeWorld applet. **Figure 1** shows my ShapeWorld applet running in the Metrowerks Java applet runner. The ten shapes all appear in red on a Canvas with a background color of yellow. The shapes are randomly distributed throughout the Canvas.

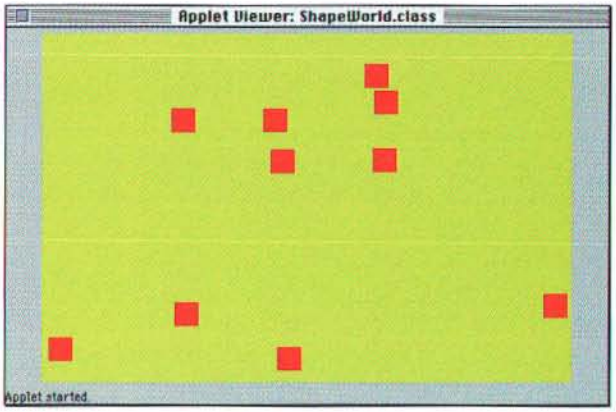


Figure 1. *The ShapeWorld applet with no shapes selected.*

Figure 2 shows what happens when I click the mouse inside a shape. In this case, I've selected six of the ten shapes. A two pixel border is used to mark a shape as selected.

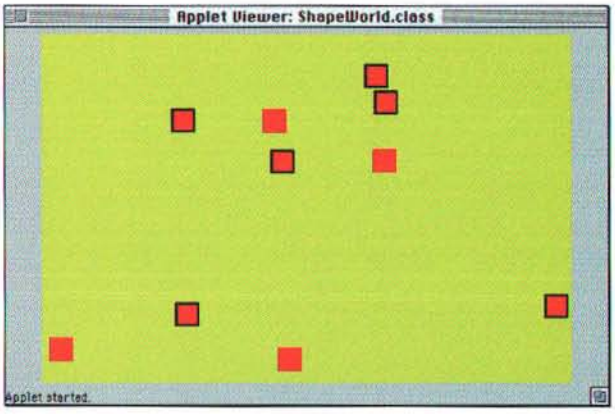


Figure 2. *The ShapeWorld applet with six shapes selected.*

The ShapeWorld Source

ShapeWorld creates four classes. ShapeWorld is the main entry point and extends the java.applet.Applet class. The ShapeCanvas class implements the Canvas. The Shape class is an abstract class which means that I won't be creating any Shape objects. The RectShape class extends the Shape class and is used to create the ten shapes you see in the ShapeWorld Canvas.

As you look through the Shape class, remember that all the variables and functions will be inherited by any classes that extend Shape. highlighted is set to true if the shape is selected, false otherwise. shapeCanvas is the enclosing Canvas. shapeX, shapeY, shapeWidth, and shapeHeight define the boundries of this Shape. boundsRect does the exact same thing, but stores this same info in a Rectangle instead of in individual variables. Why the duplication? With both forms available, I don't have to spend time converting from one form to the other. This might seem like a frivolous waste of memory, but when we are stepping through a large number of shapes, perhaps during a redraw cycle, the little time we save can make a difference.

```
import java.awt.*;
import java.util.*;

abstract class Shape
{
  boolean    highlighted;
  ShapeCanvas shapeCanvas;
  int      shapeX, shapeY, shapeWidth, shapeHeight;
  Rectangle  boundsRect;
```

The Shape constructor initializes the Shape variables. The draw() function is declared abstract, which means it *must* be overridden by any class that extends Shape. Note that an abstract function doesn't have a function body. setHighlight() sets highlighted to a new value. isHighlighted() returns the value of highlighted. Finally, isPointInShape() takes advantage of the Rectangle classes' inside() function to tell you if the specified point is inside the Shape's bounding Rectangle.

```
  Shape( ShapeCanvas canv, int x, int y,
                int width, int height )
  {
    shapeCanvas = canv;
    highlighted = false;

    shapeX = x;
    shapeY = y;
    shapeWidth = width;
    shapeHeight = height;

    boundsRect = new Rectangle( x, y, width, height );
  }

  abstract public void draw( Graphics g );

  public void setHighlight( boolean newHighlight )
  {
    highlighted = newHighlight;
  }

  public boolean isHighlighted()
  {
    return highlighted;
  }

  public boolean isPointInShape( int x, int y )
  {
    return boundsRect.inside( x, y );
  }
}
```

RectShape extends the Shape class. The RectShape constructor just passes on its parameters to its superclass constructor, which is the Shape constructor. If the shape is selected, the draw() function draws a black rectangle, insets the rectangle by two pixels on each side, then draws a red rectangle. I could have used two calls to drawRect() to frame the black rectangle followed by a call to fillRect() to fill in the red, but this made the code easier to read.

```
class RectShape extends Shape
{
  RectShape( ShapeCanvas canv, int x, int y,
                  int width, int height )
  {
    super( canv, x, y, width, height );
  }

  public void draw( Graphics g )
  {
    if ( isHighlighted() )
    {
      g.setColor( Color.black );
      g.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
      g.setColor( Color.red );
      g.fillRect( shapeX+2, shapeY+2,
                  shapeWidth-4, shapeHeight-4 );
    }
    else
    {
      g.setColor( Color.red );
      g.fillRect( shapeX, shapeY, shapeWidth, shapeHeight );
    }
  }
}
```

The ShapeCanvas class is my main drawing area. The shapes variable is of the java.util.Vector class. A Vector implements a growable array of objects. I can access the objects via random access (using an integer index) or by stepping through the list. Since I want to tie a list of shapes to my Canvas, a Vector is a natural structure in which to store references to the Shapes. Spend some time reading the java.util.Vector doc to truly appreciate the value of the Vector. In addition, you should read about the Enumeration interface (on the java.util page) that the Vector class implements. The Enumeration interface allows you to step through your Vector using the functions hasMoreElements() and nextElement().

```
class ShapeCanvas extends Canvas
{
  Vector   shapes;
  Shape    curShape;
```

The ShapeCanvas() constructor creates a new Vector and sets curShape to null. I won't make use of curShape this month, but I will use it next month to keep track of the last selected Shape. The constructor also sets the background color and changes the Canvas' size.

```
  ShapeCanvas( int width, int height )
  {
    shapes = new Vector();
    curShape = null;

    setBackground( Color.yellow );

    resize( width, height );
  }
```

addShape() adds a shape to the Vector. paint() steps through the shapes Vector, retrieving the current Shape and calling its draw() method. This method of stepping through a Vector is extremely useful. Note that nextElement() returns an Object, so I needed to cast the returned value to Shape. By retrieving a Shape instead of a specific derived class such as RectShape(), I can use this code to draw any Shapes stored in the list, no matter the type, as long as each shape is derived from Shape.

```
  public void addShape( Shape newShape )
  {
    shapes.addElement( newShape );
  }

  public void paint( Graphics g )
  {
    for ( Enumeration e = shapes.elements();
                e.hasMoreElements(); )
    {
      Shape s = (Shape)e.nextElement();
      s.draw( g );
    }
  }
```

findInShapeList() also steps through the shapes Vector. In each case, I call each Shape's isPointInShape() function to see if the specified point is in the current Shape. If the point *was* in the Shape, I flip highlighted and redraw the Shape (since it is now selected.)

```
  public Shape findInShapeList( int x, int y )
  {
    for ( Enumeration e = shapes.elements();
                e.hasMoreElements(); )
    {
      Shape s = (Shape)e.nextElement();

      if ( s.isPointInShape( x, y ) )
      {
        s.setHighlight( ! s.isHighlighted() );
        s.draw( getGraphics() );

        return s;
      }
    }

    return null;
  }
```

The update() function calls paint(). The default update() (inherited from the Component class) erases the area to be updated and then calls paint(). By calling paint() without the erase, I avoid an annoying flicker. This trick is well worth remembering. To truly understand what it does, comment out the entire update() method (not just the call to paint() which will prevent drawing from occuring at all!!!)

```
  public void update (Graphics g)
  {
    paint(g);
  }
```

The mouseDown() function gets called when I click the mouse in the Canvas. At the moment, I don't do anything interesting with the Shape returned by findInShapeList(), but I will next month.

Mac, go out and touch the world for only $199!

ADB I/O

ADB I/O lets your customers' Macs control things, it lets them feel.
ADB I/O lets the Mac be part of the physical world.

Thousands of Uses

Science, Multimedia, Children's Museums, Home Automation, Theatre Stages, Industrial Testing, Medical Research, Bonsai Watering, Robotics, Weather Stations—anything that can be electronically measured or controlled can use the ADB I/O.

No Serial Ports Occupied

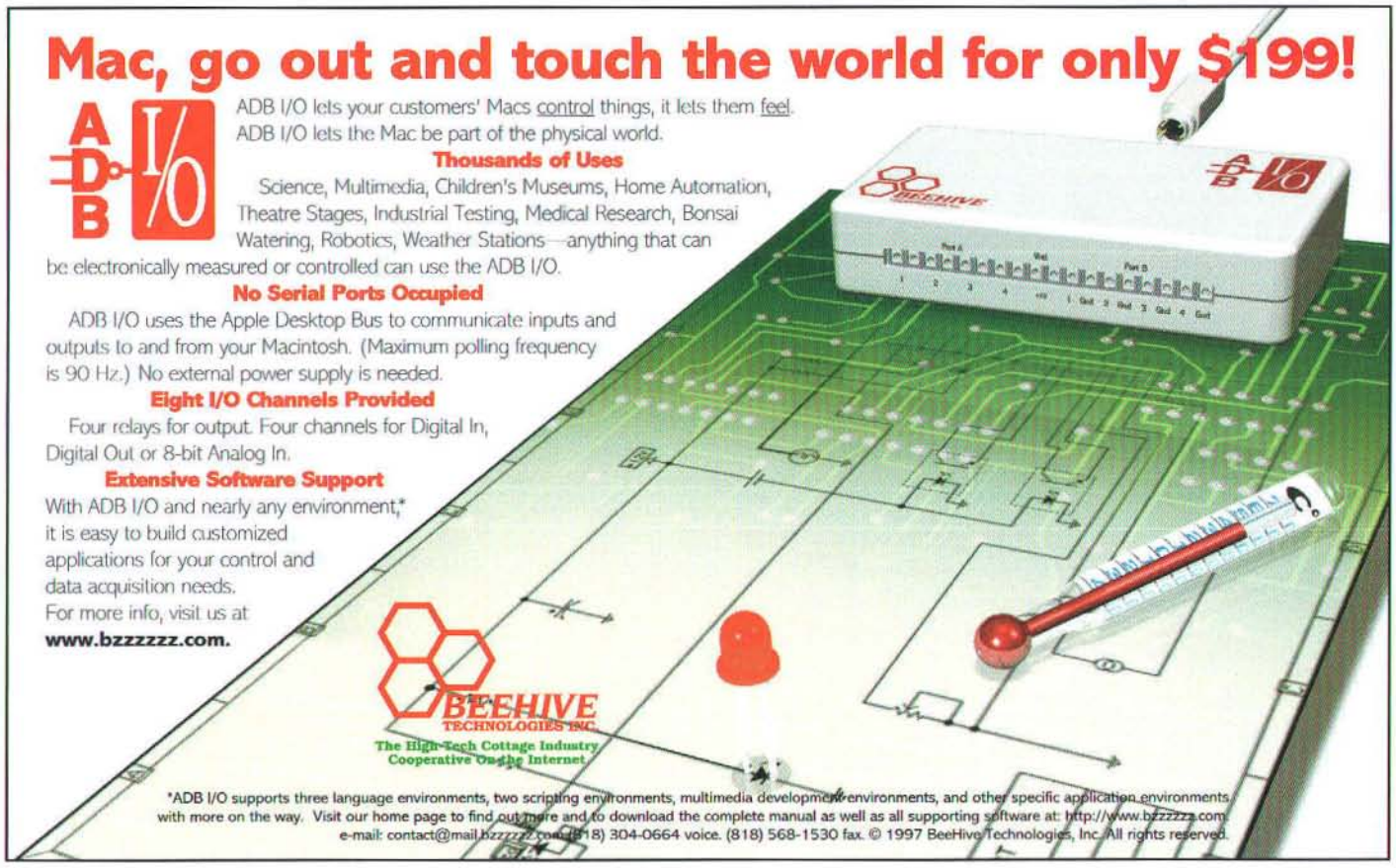
ADB I/O uses the Apple Desktop Bus to communicate inputs and outputs to and from your Macintosh. (Maximum polling frequency is 90 Hz.) No external power supply is needed.

Eight I/O Channels Provided

Four relays for output. Four channels for Digital In, Digital Out or 8-bit Analog In.

Extensive Software Support

With ADB I/O and nearly any environment,* it is easy to build customized applications for your control and data acquisition needs.
For more info, visit us at
www.bzzzzzz.com.



BEEHIVE TECHNOLOGIES INC.
The High-Tech Cottage Industry Cooperative On the Internet

*ADB I/O supports three language environments, two scripting environments, multimedia development environments, and other specific application environments, with more on the way. Visit our home page to find out more and to download the complete manual as well as all supporting software at: http://www.bzzzzzz.com
e-mail: contact@mail.bzzzzzz.com (818) 304-0664 voice. (818) 568-1530 fax. © 1997 BeeHive Technologies, Inc. All rights reserved.


```
  public boolean mouseDown( Event e, int x, int y )
  {
    curShape = findInShapeList( x, y );

    return true;
  }
}
```

The last thing I want to point out this month is the use of the Random class. ShapeWorld creates 10 new **RectShapes** (remember, you can't actually create a Shape since you can't create an instance of an abstract class) using the random values returned by **Random()**. Random is described on the page java.util.Random. Be sure to check out that page of documentation, as it describes a variety of random number distributions and returned types.

```
public class ShapeWorld extends java.applet.Applet
{
  ShapeCanvas sCanvas;
  final int   shapeWidth = 20;
  final int   shapeHeight = 20;

  public void init()
  {
    int       x, y;

    sCanvas = new ShapeCanvas( 440, 290 );
    add( sCanvas );

    Random ran = new Random();
    Rectangle b = sCanvas.bounds();

    for ( int i=1; i<=10; i++ )
    {
      x = b.x + (int)((float)(b.width) * ran.nextFloat() );
      if ( x > b.x + b.width - shapeWidth )
        x -= shapeWidth;

      y = b.y + (int)((float)(b.height) * ran.nextFloat() );
      if ( y > b.y + b.height - shapeHeight )
        y -= shapeHeight;

      RectShape r = new RectShape( sCanvas, x, y,
                shapeWidth, shapeHeight );
      sCanvas.addShape( r );
    }
  }
}
```

Till Next Month...

As promised, next month I'll add some new functionality to ShapeWorld. Before you peek at next month's column, try your hand at adding a new Shape subclass to ShapeWorld. Experiment. Read the doc. I'll see you then... MT

By Jeremy Vineyard, Viperware

Sprockets are Forever

Macs Were Made to Play Games

The Beginning...

When I was a child, I used to beg my parents for quarters to play in the local arcade. Wasting thousands of hours with this activity, I developed a passion for creating games. The desire to create my own games magnified with every quarter I spent.

Reality is often harsh, especially for the would-be game developer. I discovered that games can be very difficult to write. A good game developer must have extensive knowledge of animation techniques, music composition, sound recording, graphics synchronization, algorithm design, data structures, and more. Game developers lose sleep over the slowdown caused by CD-quality sound and full-screen scrolling graphics, and the complexities of an arcade game type environment. Until fairly recently, there wasn't much documentation explaining how to write games, if you wanted to learn, you did so by trial and error.

Not long ago, Apple officially announced something that Mac lovers have known for years: the Mac is a game machine. In fact, the Mac is a great game machine, featuring built-in support for high-resolution graphics and CD-quality sound. Along with this announcement, Apple resolved to help Mac game developers write even cooler games for the Mac. The result — Apple's Game Sprockets.

What is a Sprocket?

A sprocket is a component of Apple's new game development architecture. Each sprocket adds extra features to the MacOS that specifically benefit those who are writing games. There are six sprockets currently under development by Apple: DrawSprocket, SoundSprocket, InputSprocket, NetSprocket, SpeechSprocket, and QuickDraw 3D Rave. SpeechSprocket is just a repackaged version of Apple's Speech Recognition Manager. Game Sprockets will run only on PowerMacs, but certain sprockets may be ported to 68k in the future. Game Sprockets require the newest system software including OpenTransport 1.1, Display Manager 2.0, and Sound Manager 3.2.1.

DrawSprocket

Prepare for battle! Flames rupture your view; sirens wail. You have the enemy in your sights. Fire! The screen jerks and the action grinds to a halt. Unexpectedly, you find yourself waiting for the screen to refresh. As the computer calculates the next frame of animation, your senses are brought to terms with reality. You lose interest, quit, and go to your living room to read a book. Full-screen, high-resolution graphics are quite possibly the bane of all computer games.

The Macs make great game machines, but they lack certain characteristics that make the primeval PCs more suitable for playing games. One noticeable shortcoming is that unlike PC's, most Macs don't have page-flipping. Page-flipping is the ability to almost instantly switch frames of animation by simply telling the display device (the monitor) to look somewhere else in memory for the display data (the graphics). DrawSprocket provides transparent support for page-flipping on the Macs that support it, and falls back on more compatible methods for those that don't. When page-flipping is not available, DrawSprocket reverts to using CopyBits, which is much slower than page-flipping because QuickDraw must copy the entire contents of each frame of animation onto the screen.

DrawSprocket also supports multi-buffering, with which the next frame of animation can be drawn even while the current

Jeremy Vineyard is a senior software engineer at Viperware, a Mac game company. His interests include developing games, software utilities, and application class libraries. He can be reached at <jeremyv@viperware.com>.

frame is being copied to the screen. Multi-buffering can decrease the animation bottleneck that occurs when the processor overtakes the speed of the display device.

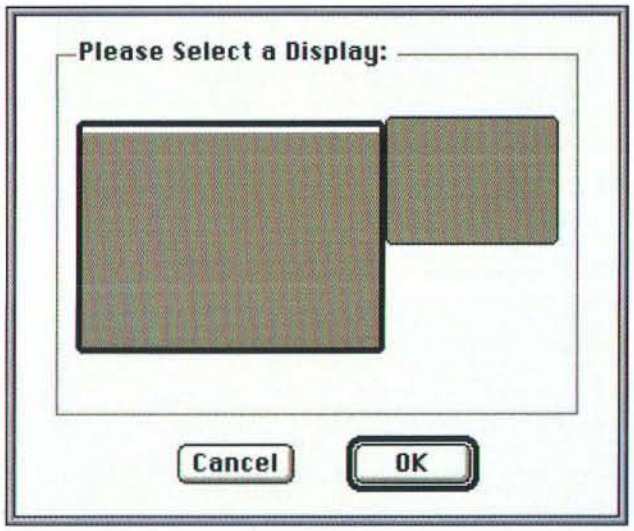


Figure 1. *Select a Display with DrawSprocket.*

DrawSprocket offers common game-related tasks such as a user interface for selecting monitors and providing an easy way for games to take control of the entire screen, which includes hiding the menu bar and blanking all unused monitors. DrawSprocket takes care of fading the screen to any color by altering a monitor's gamma tables. This is good for making a cool fade, setting up the screen for the next scene, and fading back into the game.

DrawSprocket makes it easy for you to include an underlay in your game. An underlay is an image that is always behind all other graphical elements. For example, you might have a scrolling picture of a mountain that is always in the background. In the future, DrawSprocket will support overlays allowing game developers to easily implement foreground images.

DrawSprocket's routines can provide your game with a consistent frame rate so that the animations don't change speed when they are becoming more or less processor-intensive. For the animated elements, Apple provides a sprite manager within QuickTime.

InputSprocket

Have you been frustrated while trying to use a Mac joystick? Until the introduction of InputSprocket, the Mac had no built-in support for game input devices, resulting in emulated performance by the Mac joysticks, flight sticks, and game pads.

InputSprocket is an API for the Mac joystick developers that provides a consistent user experience when using joysticks and other game-related input devices on the Mac. Joystick manufacturers write drivers for their devices that communicate with games about the elements of the joystick. Some examples of elements are buttons, controllers, directional pads, and others that are common to game-related input devices. Games that use InputSprocket can interact directly with these elements, eliminating the need for the elements to be emulated through the mouse or keyboard.

You can create your own interface for choosing the active game device and assigning functions to game controls, but InputSprocket provides a standard user interface for you.

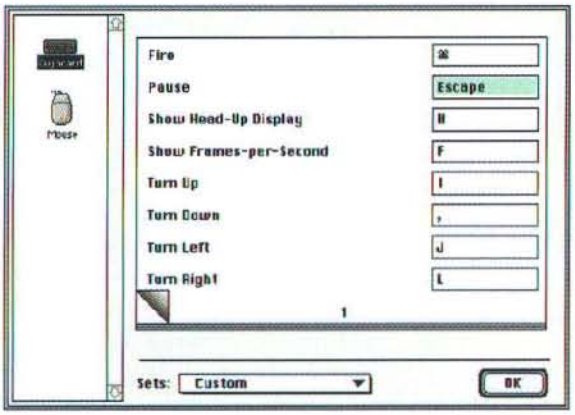


Figure 2. *InputSprocket Interface.*

For a game input device to work with InputSprocket, a driver must be written for that device. As of the writing of this article, no joystick manufacturers shipped InputSprocket drivers for their products, but companies such as Gravis and Thrustmaster have demonstrated prototype InputSprocket drivers. InputSprocket has problems working with some third-party input devices such as Kensington mice, but Apple developers are working to fix this problem.

QuickDraw 3D RAVE

QuickDraw 3D RAVE (Rendering Acceleration Virtual Engine) is a Hardware Abstraction Layer (HAL) that allows game developers who write their own 3D rendering engines (as does Bungie for Marathon) to take advantage of the features offered by 3D accelerator cards when they are present. QuickDraw 3D Rave is a stripped-down version of QuickDraw 3D, and it is highly optimized, making it ideal for game developers who need every amount of speed they can get.

Games communicate with QuickDraw 3D RAVE by converting their rendered images into triangles, which RAVE then sends to the accelerator card if present, or otherwise emulates in software. In addition to supporting 3D accelerator cards, RAVE will soon be cross-platform, allowing game developers to easily port their low-level 3D rendering code.

SoundSprocket

3D games are the rage nowadays. This is evident by the huge sales of games like Doom and Marathon. Along with freedom of movement comes the responsibility to make the user experience as realistic as possible. Players are expecting to

be drawn into their games more and more as computer processing power increases. 3D sound adds an extra dimension of reality and helps the game developer write really cool games.



Figure 3. *Configuring a Sound Device for 3D Sound.*

SoundSprocket adds the capability of making sounds appear to come from positions relative to the game player. This allows realistic feedback such as: "The aliens are to my left!" and can make game play much more enjoyable. Like all the other sprockets, SoundSprocket provides a standard user interface for configuring sound output devices to take maximum advantage of their 3D sound capabilities.

Future versions of SoundSprocket promise to add effects such as altering sound as it travels through the environment. Sound will be heard differently based on the humidity of the environment, and will even be able to travel through dense fog or water. Sound will also reverberate by bouncing off walls.

You don't have to have a 3D game to benefit from 3D sound. Most games that use sound effects can benefit from more realistic sound and a more in-depth user experience.

NetSprocket

As adrenaline pulses through your veins, your finger slips from sweat as you pull at the trigger. The force from the rocket blast kicks you back. Splat! A direct hit. You drool in ecstasy as your opponent's corpse splatters to the ground. Are you competing in Bloodsport? No- you have just finished off the last of your assailants in a networked game of Marathon. Unlike competing against a personality-less computer, going head-to-head with your best friends (or even your enemies) can be a heck of a good time.

Macs have been one of the easiest platforms to network together since their introduction. Setting up a local area network is as simple as connecting a cable between two Macs. This ease of use takes most of the pain out of adding networking to Mac games. However, because there are many different protocols that a network might support (IP, IPX, AppleTalk, Serial, etc.), a game developer probably won't have time to implement all of these protocols in addition to testing the reliability of their network code.

NetSprocket provides a set of high-level routines designed specifically to help developers add networking code to their games. NetSprocket hides the low-level details of the network from the programmer and takes care of many tasks that the developer would otherwise be forced to implement.

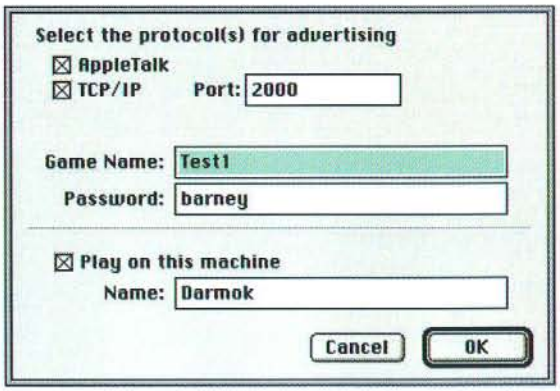


Figure 4. *Hosting a Network Game.*



Figure 5. *Joining a Network Game.*

NetSprocket is based on a client/server architecture in that one player acting as a server hosts a game session, and other players join as clients to the network game. In the future, NetSprocket will provide more models other than the client/server model for interacting between networked players. NetSprocket hides the implementation details of supporting various protocols, saving the developer time needed to re-write the networking code for each protocol. NetSprocket is fast and fault tolerant — when one computer in the networked game crashes, it doesn't bring down everyone else. Since the low-level network code is already written, NetSprocket can save a game developer hundreds of hours of work.

NetSprocket assigns each player a player ID and a player type. Player types might include referees, observers, and participants. Players can belong to groups (red team, black team, etc.).

A networked game works like a conversation — players pass messages back and forth. This communication activity is known as a **protocol**. A network protocol is very much like a conversation — players ask questions and wait for responses. Although NetSprocket makes it very easy to pass messages between networked players, knowledge of designing protocols is important. Fortunately, designing protocols is not a difficult topic to learn, and can end up being somewhat addicting. (Or maybe it's just me?)

To Infinity and Beyond

Apple is not supporting 68k machines with its Game Sprockets architecture. Until Apple develops 68k versions of Game Sprockets, Game Sprocket developers must limit their audience to PowerMac users.

Apple has a tarnished reputation for settling for complacency when it releases new technologies to developers (see QuickDraw GX, AOCE). In order for Game Sprockets to thrive, Apple must provide game developers with an abundance of tools, source code, and updates to the existing API. Thankfully, Apple has been doing a good job with Game Sprockets, keeping updates coming steadily and keeping in touch with the developers who use the technologies.

Apple's Game Sprockets are definitely a step in the right direction if they want to remain a competitive OS solution into the future. I look forward to seeing games released that take advantage of the features that are offered. Thank you to those people within Apple who are working hard to bring more cool games to the Mac!

Footnote

The Internet is a great place to find information on Apple's Game Sprocket architecture. Here are some URL's that can help you get started.

Apple's Game Sprocket Page
<http://dev.info.apple.com/evangelism/games/games.html>

Developer University Course- Game Development with Sprockets
<http://devworld.apple.com/dev/du/Using_Game_Sprockets/Using_Sprockets.html>

Licorice Web Page for Game Sprocket Developers
<http://www.viperware.com/licorice> MT

Visit MacTech Magazine's Web site!
http://www.mactech.com



PROGRAMMER'S CHALLENGE



By Bob Boonstra, Westford, MA

OTHELLO™

This month's Challenge is going to be another round-robin competition for a well-known board game — this time the game of Othello. The classic game of Othello is played on an 8x8 board using discs that are black on one side and white on the other side. The game starts with four discs in the center squares of the board, two black discs on diagonally adjacent squares, and two white discs on the other diagonally adjacent squares. Players alternate placing an additional disc, with black moving first. A move consists of "outflanking" one or more of your opponent's discs. Outflanking means placing a disc so that at least one row of your opponent's discs is bordered by a disc of your color, including the disc just placed on the board. A row is defined as one or more discs of a single color in a continuous straight horizontal, vertical, or diagonal line. When a player moves, the row or rows of outflanked discs are flipped over showing his or her color. If a player cannot outflank a disc, the turn is forfeited and the opponent takes another turn. The game is over when the board is filled or neither player can move. The player with the most discs showing is the winner.

In this Challenge, the game will be generalized to boards larger than 8x8. The prototype for the code you should write is:

```
Boolean /*legalMove*/ Othello (
  long boardSize,       /* number of rows/columns in the game board */
  long oppRow,          /* row where opponent last moved, 0 .. boardSize-1 */
  long oppCol,          /* column where opponent last moved, 0 .. boardSize-1 */
  long *moveRow,        /* return your move - row, 0 .. boardSize-1 */
  long *moveCol,        /* return your move - column, 0 .. boardSize-1 */
  void *privStorage,    /* preallocated storage for your use */
  long storageSize,     /* size of privStorage in bytes */
  Boolean newGame,      /* TRUE if this is your first move in a new game */
  Boolean playBlack     /* TRUE if you play first (black pieces) */
);
```

For your first move, Othello will be called with newGame set to TRUE. The size of the board, an even number between 8 and 64, will be provided in boardSize. On your first move you should initialize the board with white tiles at (row,col) = (boardSize/2-1,boardSize/2-1) and (boardSize/2,boardSize/2), and black tiles at (boardSize/2-1,boardSize/2) and (boardSize/2,boardSize/2-1). Rows and columns are numbered from 0 to boardSize-1. If playBlack is TRUE, you are to play the black pieces, and therefore play first. Otherwise, you play the white pieces. Your code and the code of an opponent will alternate play. Your opponent's move will be provided in (oppRow, oppCol), which will be set to (-1,-1) if your opponent is unable to move or if you are moving first. When your code is called you should flip your tiles that were outflanked by your opponent's move, calculate your own move, and store it in (*moveRow, *moveCol). If you are unable to move, store the values (-1,-1). If for any reason you believe your opponent has made an illegal move, Othello should return a value of FALSE, otherwise it should return TRUE.

Your code will be provided with storageSize bytes of preallocated storage (at least 1 MB) pointed to by privStorage. This storage will be pre-initialized to zero before your first move and will persist between moves. You should not allocate any additional dynamic storage beyond that provided by privStorage. Small amounts of static storage are permitted.

The Challenge will be scored by a tournament where each entry plays against each other entry twice for each of a number of board sizes, once playing the black pieces and once playing the white pieces. In the event that a large number of entries are received, another fair tournament schedule may be used. The score will be based on the margin of victory (or loss) and the

THE RULES

Here's how it works: each month we present a new programming challenge. First, write some code that solves the challenge. Second, optimize your code (a lot). Then, submit your solution to MacTech Magazine. We choose a winner based on code correctness, speed, size, and elegance (in that order of importance) as well as the submission date. In the event of multiple equally desirable solutions, we'll choose one winner (with honorable mention, but no prize, given to the runner up). The prize for each month's best solution is a $100 credit for Developer Depot™ and a limited-edition "The Winner! MacTech Programmers Challenge" T-shirt (not available in stores anywhere).

Unless stated otherwise in the problem statement, the following rules apply: All solutions must be in ANSI compatible C or C++, or in Pascal. We disqualify entries with any assembly in them (except for challenges specifically stating otherwise.) You may call any Macintosh Toolbox routine (e.g., it doesn't matter if you use NewPtr instead of malloc). We compile all entries into native PowerPC code with compiler options set to enable all available speed optimizations. The development environment to be used for selecting the winner will be stated in the problem. **Limit your code to 60 characters per line** or compress and binhex the solution; this helps with e-mail gateways and page layout.

We publish the solution and winners for each month's Programmer's Challenge three months later. All submissions must be received by the 1st day of the month printed on the front cover of this issue.

You can get a head start on the Challenge by reading the Programmer's Challenge mailing list. It will be posted to the list on or before the 12th of the preceding month. To join, send an email to listserve@listmail.xplain.com with the message "subscribe challenge-A".

Mark solutions "Attn: Programmer's Challenge Solution" and send it by e-mail to one of the Programmer's Challenge addresses in the "How to Communicate With Us" section on page 2 of this issue. Include the solution, all related files, and your contact info.

MacTech Magazine reserves the right to publish any solution entered in the Programmer's Challenge. Authors grant MacTech Magazine the exclusive right to publish entries without limitation upon submission of each entry. Authors retain copyrights for the code.

Announcing

MICROGUARD PLUS.™

Why so many developers are switching to MicroGuard copy protection

MicroGuard is committed to uncompromising technological superiority

"Technology at its peak" is our commitment to you. That is why we have brought you **MicroGuard Plus**™. And, MicroGuard Plus is **100% backwards** compatible with MicroGuard. This means MicroGuard and MicroGuard Plus can be used interchangeably. Just as we promised!

MicroGuard has now surpassed its own technological lead, actually improving on the best

MicroGuard Plus is everything MicroGuard is, plus 40-bit encryption, two additional passwords, 64-Bit Array, 32-byte public area, 45% size reduction, enhanced counter and more. In addition, MicroGuard Plus offers two new utilities: **QuickGuard**™ and EasyGuard™ allow you to protect your applications without touching your source code. The only feature that is not plus is the price. :-)



MicroGuard offers you the most sophisticated network protection

Our network protection, **MicroGuard Net**™, is so superior, we had to hire an Apple network engineer to execute our specifications.

MicroGuard is the best selling Macintosh key in the world

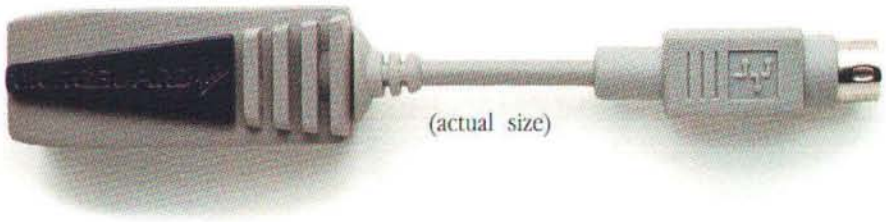
MicroGuard sells more Macintosh copy-protection keys than anyone else in the world!

MicroGuard is the only key developed by Mac developers, and is the first and only 100% ADB savvy key

We have been developing Mac applications as a seed development house since 1984. We are not a PC protection company that has come to you with a Mac product. MicroGuard is fully ADB savvy and offers extended addressing. Unlike other protection devices, MicroGuard never clashes with other keys. Only MicroGuard offers this level of sophistication.

MicroGuard delivers developer support within 24 hours

We will answer any inquiry you have within 24-hours. We also have a fully loaded AppleLink bulletin board which contains all our libraries, tech notes, Q&A and nearly everything you'll ever need!



For more information and to order a Developer's Kit or to receive a free CD ROM about MicroGuard, please contact us at:

MicroGuard USA: Tel: (303) 320-1628 • Fax: (303) 320-1599 • AppleLink: M.GUARD
International: Tel: (972) 3 558-2345 • Fax: (972) 3 558-2344 • AppleLink: MICROGUARD

Mac™OS

Pirate Busters

execution time used to compute the moves. For each game, the score will be computed by

[(# of player's pieces showing - # of opponent's pieces showing) - (execution time in seconds)/30] /

(boardSize*boardSize)

The player with the highest score for all games in the tournament will be the winner.

Those of you needing more information about the rules of the game can check out your local toy store, or look at <http://www.daimi.aau.dk/~tusk/pbmserv/othello/othello.rules.html>. Other information can be found at <http://www.armory.com/~iioa/othguide.html>, the International Internet Othello Association page.

Othello is a registered trademark of Tsukuda Original, licensed by Anjar Co., copyright 1973, 1990 by Pressman Toy Corporation.

Three Months Ago Winner

Congratulations to **Thomas Studer** (Syracuse, N.Y.) for submitting the winning entry to the Router Rules Challenge. The problem was to generate a set of (mask, value, allow/deny) triplets that could be used by a router to allow net access to a specified set of subnet addresses. Given a subnet address, the rules would be scanned in sequence by the router, and the first rule to fire would determine whether access was allowed or denied. A rule fires when the subnet address, logically ANDed with the rule mask, is equal to the rule value. Solutions were to minimize a score that was the sum of two quantities, the number of rules generated, and the time taken to generate those rules (in half-seconds).

Of the six solutions I received, only two of them worked correctly for my test cases. The other four either generated rules that produced incorrect results for some input subnet values or had not completed execution after running overnight. Thomas' winning solution assigns each input value to a group determined by the number of bits set in each "chunk" of 1, 2, 4, or 8 bits. The code then uses this mapping to search for "buddy" input values that differ in only one bit position. When such a "buddy" value is found, the values are combined, and a mask is updated to indicate which bits should be masked out in a router rule. The code makes use of three large pre-built BitGrpMapper tables that allow Thomas to calculate the number of bits set in each chunk of 2, 4, or 8 bits. For each chunk of size n, these tables represent the number of bits set in each input value as a base n+1 number, an interesting and compact representation.

The second-place solution by **Alan Hart** used a recursive technique to generate the Karnough map representing the allowed values. While this approach generated more rules (and therefore a poorer score) than the winning solution, it ordered the rules more optimally, in that rules governing a larger number of allowed subnets occurred earlier. This was not a criterion for scoring this Challenge, but it would be an important real-world consideration.

More compact rule sets than those generated by the two correct solutions might be possible. The very long-running solution mentioned above appeared to generate a small number of rules, half as many in some cases, at significant execution time expense.

The table below provides the language, code size, and data size for each of the six entries received. For the two correct entries, it also provides the score (based on rules generated and execution time), the average number of rules scanned before an allowed subnet value is granted access (summed over all test cases), and the total number of rules generated for all test cases.

Name	Language	Score	Avg Rules	Rules Gen	Code	Data
Thomas Studer	C++	3365	1414	3365	3844	12824
Alan Hart	C	4716	1193	4715	1472	148
E. M.	C++	*			6308	6672
L. N.	C	*			4024	220
K. S.	C	*			2472	8
M. Y.	C++	*			3312	299

TOP 20 CONTESTANTS

Here are the Top 20 Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points
1.	Munter, Ernst	195
2.	Gregg, Xan	114
3.	Larsson, Gustav	87
4.	Lengyel, Eric	40
5.	Lewis, Peter	32
6.	Boring, Randy	27
7.	Cooper, Greg	27
8.	Antoniewicz, Andy	24
9.	Beith, Gary	24
10.	Kasparian, Raffi	22
11.	Cutts, Kevin	21
12.	Nicolle, Ludovic	21
13.	Picao, Miguel Cruz	21
14.	Brown, Jorg	20
15.	Gundrum, Eric	20
16.	Studer, Thomas	20
17.	[Name deleted]	20
18.	Karsh, Bill	19
19.	Mallett, Jeff	17
20.	Nevard, John	17

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place .....20 points
2nd place.....10 points
3rd place.....7 points
4th place.....4 points
5th place.....2 points
finding bug.....2 points
suggesting Challenge...2 points

Here is Thomas' winning solution:

RouterRules.cp

Copyright © 1996 Thomas Studer

```
// Memory requirements
// ———————————
// About 12K of static tables. A variable amount of heap memory. The program will run
// faster and/or yield compacter results given more memory (in most cases). The
// maximum amount of dynamic memory is approximately (chunkSize+1)^chunkCount
// times the size of a pointer + the number of input values times 12 bytes (the size of
// BitGrpEntry). 'chunkSize' is 1, 2, 4 or 8. 'chunkCount' is the width (in bits) of the
// input values / chunkSize, rounded up. 'chunkCount' is between 4 and 32, inclusive.
//
// How it works
// ————————
// - The values to be reduced are arranged in memory in a number of linked lists. The
// program then loops through these lists trying to find values (with matching masks)
// that differ in exactly one bit ('buddy' values). The value in a pair of such compatible
// values that has a '1' in the only differing bit position is thrown away and the mask of
// the the remaining value is updated by clearing that bit. If, for a given value no
// 'buddy' can be found, it is output and the program continues until all the internal
// lists are empty.
//
// - The values are arranged in 'bit groups' to quickly locate a given value's buddy. A bit
// group is identified by a number whose individual digits denote the number of bits set
// to 1 within a range of consecutive bits in the input value. A range of consecutive bits
// is called a 'chunk'. Chunk sizes can be 1, 2, 4 or 8. For example, an input value of 16
// bits that looks like this (chunkSize 4): 1101 1001 0001 0111, will have group value
// 3213 (base 5, since every digit denotes the number of 1's (0..4) per bit sequence
// (chunk). During initialization, all the input values are categorized using that scheme
// by storing them in lists (one list for every bit group). Pointers to the first elements
// (BitGrpEntry's) of these lists are kept in the gBitGrpLists array. A particular bit group
// can now be accessed using the bit group value as an index into that array. For any
// given value, to locate a buddy, only those bit group lists have to be searched that
// differ by 1 in exactly one digit.
//
// - The class BitGrpMapper is responsible for the initial categorization of the input
// values (through one of three lookup tables, depending on chunkSize). The class also
// calculates and stores some other values pertaining to the current run's chunk size.
// The bulk of the base 3, 5 or 9 arithmetic (namely, when for a given value the buddy
// groups have to be located) is done in RemoveLoop(). For the chunkSize == 1 case,
// no lookup table is required because the bit group number is a binary number. Some
// of the functions have been optimized for the 1 bit case.
//
// I think the algorithm is quite nice. However, there is some room for improvement in
// the implementation. Moreover, the style of the code could be improved - it is not
// particularly readable and it doesn't make enough use of types to make the code more
// expresive (basically another one of those C turned C++ programs - I'm working on it).
// ———————————————————————————————

#define ALTER_INPUT_VALUES  1

#include "ProvidedCode.h"
#include "BitGrpMapper.h"

// Data structs and types

enum ErrCode {    kNoErr = 0,
                  kErr = 1
};

struct BitGrpEntry {
  BitGrpEntry     *next;
  ulng            value;
  ulng            mask;
};

typedef BitGrpEntry *BitGrpEntryPtr;
```

Prototypes

```
ErrCode  Init( void );

ErrCode Process( void );

long    CleanUp( void );

void    MakeComplement(    void );

void    InitBitGrpLists(   long    startValue,
                           long    pastValue );

void    ClearMemory(       long  *p,
                           long  blockCount );

void    ProcessLists(      void );

long     RemoveValues(     void );

void    RemoveLoop(        long            curIdx,
                           BitGrpEntryPtr  beforeEntry,
                           BitGrpEntryPtr  curEntry );

void    RemoveLoop1Bit(    ulng            curIdx,
                           BitGrpEntryPtr  beforeEntry,
                           BitGrpEntryPtr  curEntry );


long    ScanAndKeep(       BitGrpEntryPtr  compareEntry,
                           BitGrpEntryPtr  beforeEntry,
                           ulng            mask,
                           ulng            matchBit );

long    ScanAndRemove(     BitGrpEntryPtr  compareEntry,
                           BitGrpEntryPtr  beforeEntry,
                           ulng            mask,
                           ulng            matchBit );

inline  long  Match(       BitGrpEntryPtr  compareEntry,
                           BitGrpEntryPtr  thisEntry,
                           ulng            matchBit );

void     AddToOutput(      BitGrpEntryPtr  curEntry );
```

Global data

```
long             *gAllowedValues;
long             gNumAllowedValues;
long             gNumBits;
Rule             *gCurRule;
long             gMaxRules;
long             gRulesLeft;
long             gBlockNumAllowedValues;
long             gStartMask;
long             gAllow;

BitGrpMapper     gBitGrpMapper; // the BitGrpMapper class
BitGrpEntryPtr   *gBitGrpLists; // Array of BitGrpEntry
                                // list headers
BitGrpEntryPtr   gFirstFreeEntry;
long             gNumBitGrpBlocks;
long             gNumValuesInLists;
```

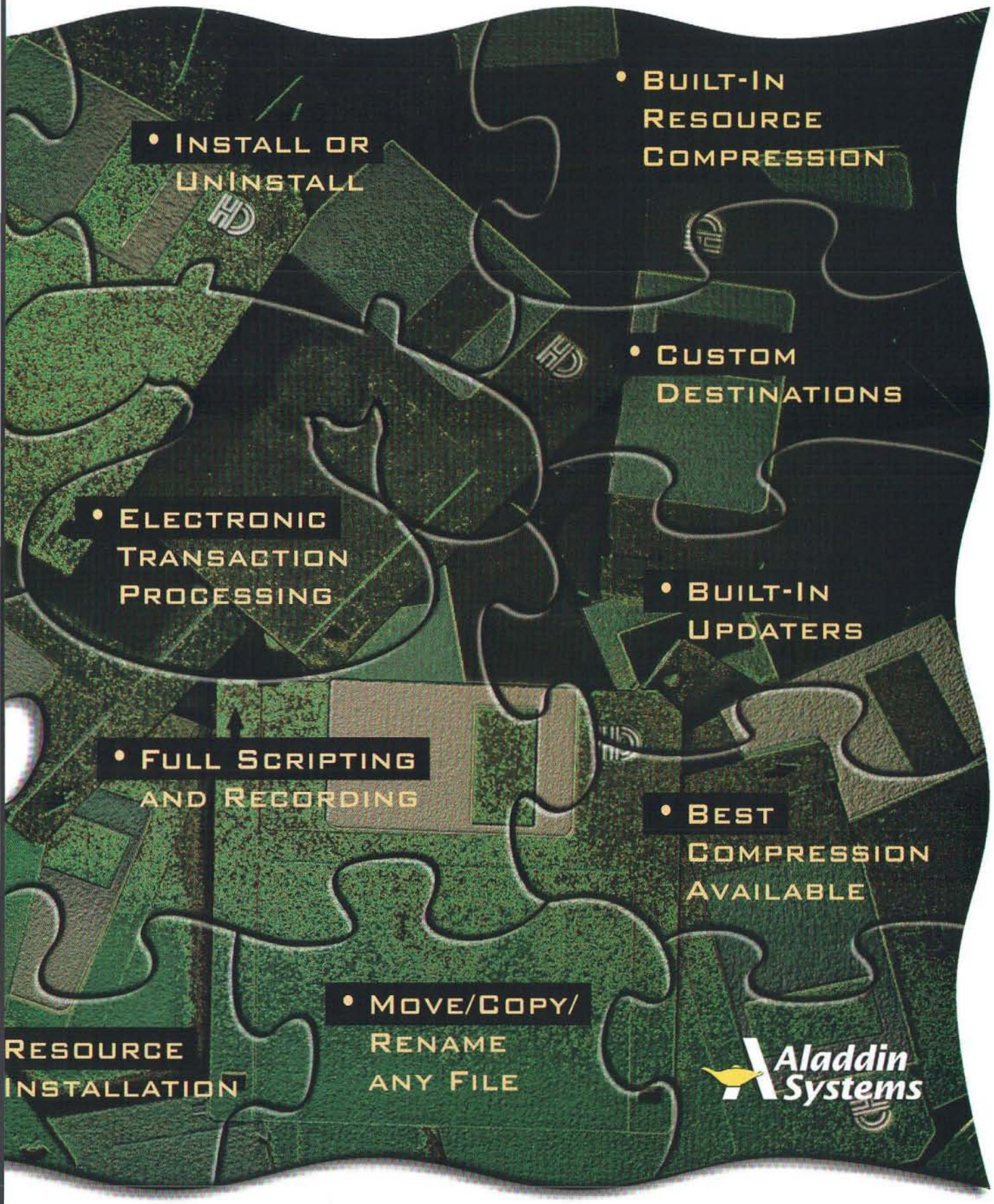
// Implementation

Init

```
// For any run, depending on the number of input values and the amount of available
// memory, various combinations of chunkSize and gBlockNumAllowedValues are
// possible, yielding different results and different execution times. There wasn't
// enough time to sufficiently analyze the program's algorithm. That's why this function
// contains a lot of guessing. The while loop in Init() starts with a small chunkSize (1 or
// 2) and tries to allocate the required amount of memory. If that fails, the number of
// input values processed at a time is split in half, requiring less memory for the actual
// values. If that still takes too much memory, the chunkSize is increased, the number
// of values to be processed is reset and the attempt to allocate memory is repeated.
//
ErrCode  Init( void )
{
  const long  kBitLimit = 27;
  const long  kMemLimit = 1L << kBitLimit;
  long        valueMem;
  long        splitCount = 0;
  long        chunkSize = 1;
```

continued on page 22


Our competitors' habit of charging for each piece of their installer program is a bit puzzling.
• CREATE UP TO 128 CUSTOM INSTALLS
• IMPROVED DEVELOPER CUSTOMIZATION
The reason we call StuffIt InstallerMaker 4.0 the Complete Installation Solution is because we include everything you need to prepare a professional package of files for installation on your user's machines. An installer, updater, and an uninstaller are just three components included for the same price one of our fine competitors charges for just the installer. And we've just added electronic transaction processing. Using our installer reduces technical support calls due to end-user errors during installation, saves disks (if distributing on floppies) or download time (if distributing on-line). You will save more than the cost of the Installer license by significantly reducing distribution costs. And the puzzle will be solved. You'll have every piece in place.
Download a free, fully-functional copy of StuffIt InstallerMaker 4.0 from www.aladdinsys.com, or call (408) 761-6200 and ask for Developer Sales.
INSTALLERMAKER 4.0
© 1996 Aladdin Systems, Inc. 165 Westridge Drive, Watsonville, CA 95076. Fax: (408) 761-6206. Internet:dev.sales@aladdinsys.com. AOL, AppleLink: ALADDIN. StuffIt InstallerMaker is a trademark of Aladdin Systems, Inc. Other products are trademarks of their respective holders.
VISIT US AT DEVELOPER CENTRAL AT MACEXPO/SF


• Install or UnInstall
• Built-In Resource Compression
• Custom Destinations
• Electronic Transaction Processing
• Built-In Updaters
• Full Scripting and Recording
• Best Compression Available
• Move/Copy/ Rename any File
Resource Installation
Aladdin Systems

continued from page 19

```
  gBitGrpLists = NULL;
  gBlockNumAllowedValues = gNumAllowedValues;
  if (gNumBits > kBitLimit) chunkSize = 2;

  while (gBlockNumAllowedValues > 0) {

  // Init gBitGrpMapper fur current chunk size
    gBitGrpMapper.Init( chunkSize, gNumBits );

    if (splitCount == 0 && chunkSize != 8 &&
        gBitGrpMapper.numGrpLists / gBlockNumAllowedValues
        > 60) {
  // Very scarce -> Force shift to next chunk size
      splitCount = 100;

    } else {

      if ( 4 * gBitGrpMapper.numGrpLists < kMemLimit) {

  // How many blocks of 8 BitGrpEntryPtr's?
        gNumBitGrpBlocks = gBitGrpMapper.numGrpLists/8 + 1;

  // How much memory for the values
        valueMem =
          gBlockNumAllowedValues * sizeof( BitGrpEntry );

        if (valueMem < kMemLimit) {

  // Allocate memory for the BitGrpEntry's
          gBitGrpLists = (BitGrpEntryPtr*)
            NewPtr(valueMem + 32 * gNumBitGrpBlocks);

  // Successful allocation?
          if (gBitGrpLists) {
            gFirstFreeEntry = (BitGrpEntryPtr)
              &gBitGrpLists[ 8 * gNumBitGrpBlocks ];
            return kNoErr;
          }
        }
      }
    }

    switch (chunkSize) {
      case 1:
        if (splitCount>=1 || gBlockNumAllowedValues < 4) {
          gBlockNumAllowedValues = gNumAllowedValues;
          splitCount = 0;
          chunkSize = 2;
          continue;
        }
      case 2:
        if (splitCount>=2 || gBlockNumAllowedValues < 4) {
          gBlockNumAllowedValues = gNumAllowedValues;
          splitCount = 0;
          chunkSize = 4;
          continue;
        }
      case 4:
        if (splitCount>=3 || gBlockNumAllowedValues < 4) {
          gBlockNumAllowedValues = gNumAllowedValues;
          splitCount = 0;
          chunkSize = 8;
          continue;
        }
    }

    gBlockNumAllowedValues /= 2;
    splitCount++;
  }

  return kErr;
}
```

MakeComplement

```
// If the number of allowed values in the input exceeds half the maximum number of
// allowed values (plus some slack), the number of values that are not in the
// gAllowedValues array are calculated and replace the values in gAllowedValues. These
// values are then to be denied.
//


void  MakeComplement( void )
{
  ulng  *bitMap;
  ulng  numLongs = (1L << gNumBits) / 32;

  bitMap = (ulng*) NewPtr( 4 * (numLongs + 8));

  if (bitMap) {

  // Clear bitMap
    ClearMemory( (long*)bitMap, (numLongs + 8) / 8 );

  // For every allowed value set its bit in bitMap
    ulng *pastVal=(ulng*)&gAllowedValues[gNumAllowedValues];
    ulng *curVal = (ulng*)gAllowedValues;

    do {
      bitMap[*curVal>>5] |= (1L << (*curVal & 0x0000001f));
      curVal++;
    } while (curVal != pastVal);

  // Determine the values to be denied by looking for
  // 0 bits in bitMap. Write them out to gAllowedValues
    ulng *curEntry = bitMap;
    ulng curIndex; // into bitMap
    ulng curBit;
    curVal = (ulng*)gAllowedValues;

    for (curIndex = 0; curIndex<numLongs; curIndex++) {
      if (*curEntry != 0xffffffff) {
        for (curBit = 0; curBit<32; curBit++) {
          if ((*curEntry & (1L << curBit)) == 0) {
            *curVal = (curIndex << 5) | curBit;
            curVal++;
          }
        }
      }
      curEntry++;
    }

  // Set the new number of values in gAllowedValues
  // and flip the gAllow variable from kAllow to kDeny
    gNumAllowedValues = curVal - (ulng*)gAllowedValues;
    gAllow = kDeny;

    DisposPtr( (char*) bitMap );
  }
}
```

InitBitGrpLists

```
// Initialization of the internal lists by reading values from gAllowedValues and storing
// them as BitGrpEntry items.
//
void  InitBitGrpLists( long    startValue,
                       long    pastValue )
{
  long           *curVal = &gAllowedValues[startValue];
  long           *pastVal = &gAllowedValues[pastValue];
  BitGrpEntryPtr  curEntry = gFirstFreeEntry;
  BitGrpEntryPtr  *curHead;

  ClearMemory( (long*)gBitGrpLists, gNumBitGrpBlocks );

// Two times the same while loop. Once for the special case of chunkSize == 1 and
// then for the general case of chunkSize == 2, 4 or 8. Only the chunkSize == 2, 4
// and 8 cases need gBitGrpMapper's LookUp method since these cases deal with
// base 3, 5 and 9 integers, respectively.

  if (gBitGrpMapper.chunkSize == 1) {
    while (curVal < pastVal) {
      gBitGrpLists[ *curVal ] = curEntry;
```

```
        curEntry->next = NULL;
        curEntry->value = *curVal;
        curEntry->mask = gStartMask;
        curEntry++;
        curVal++;
      }
    } else {
      while (curVal < pastVal) {
        curHead =
          &gBitGrpLists[ gBitGrpMapper.LookUp( *curVal ) ];
        curEntry->next = *curHead;
        curEntry->value = *curVal;
        curEntry->mask = gStartMask;
        *curHead = curEntry;
        curEntry++;
        curVal++;
      }
    }

    gNumValuesInLists = pastValue - startValue;
}
```

ClearMemory

```
// Unfortunately I don't know the PPC processors well enough to know whether the
// way this loop is unrolled really helps much.
//
void  ClearMemory(   long  *p,
                     long  blockCount )
{
  while (blockCount--) {
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
    *p = NULL;
    p++;
  }
}
```

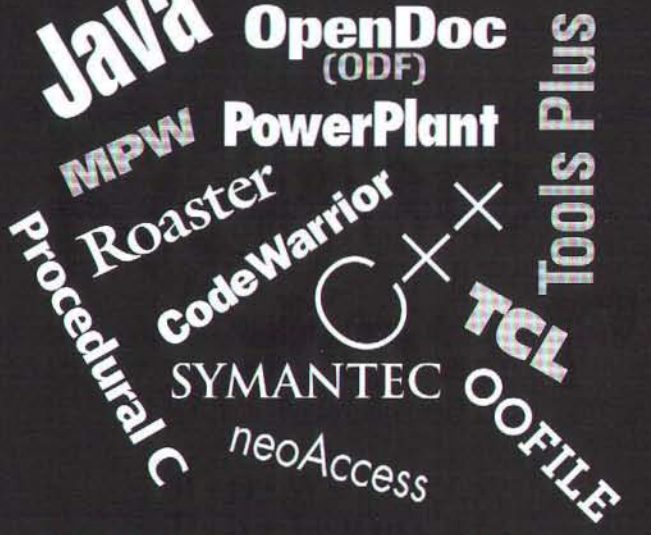
AddToOutput

```
// Add a value for which no 'buddy' can be found to the output array.
//
inline void AddToOutput(  BitGrpEntryPtr  curEntry )
{
  if (gRulesLeft) {
    // Add to output rules
    gCurRule->value = curEntry->value;
    gCurRule->mask = curEntry->mask;
    gCurRule->allow = gAllow;
    gCurRule++;
    gRulesLeft--;
  }
}
```

Process

```
// Entry point for the main processing loop. If there is enough memory, all the available
// values are considered at the same time. If memory is low, the input values are
// processed in blocks of size gBlockNumAllowedValues (likely to produce a higer
// number of output rules).
//
ErrCode Process( void)
{
  long  numValuesLeft = gNumAllowedValues;
  long  startValue = 0;
  long  pastValue = 0;
  while (numValuesLeft) {
        pastValue += gBlockNumAllowedValues;
```



AppMaker supports most popular languages and frameworks. Just point and click to design your user interface, then let AppMaker create resources and generate "human, professional quality code" to implement your design.

Use AppMaker as a prototyping and productivity tool or use it as a learning tool for Mac programming techniques or for new environments such as OpenDoc, Java, or PowerPlant.

AppMaker is just $299 and includes a one-year subscription on CD.

B • O • W • E • R • S
Development

603-863-0945
bowersdev@aol.com
http://members.aol.com/bowersdev

```
    if (pastValue > gNumAllowedValues) {
      pastValue = gNumAllowedValues;
    }
    InitBitGrpLists( startValue, pastValue );
    ProcessLists();
    if (gRulesLeft <= 0) return kErr; // Output array full
    numValuesLeft -= pastValue - startValue;
    startValue = pastValue;
  }

  return kNoErr;
}
```

ProcessLists

```
// Loop over gBitGrpLists while there are values to combine.
//
void ProcessLists( void )
{
  while (gNumValuesInLists) {
    gNumValuesInLists -= RemoveValues();
  }
}
```

RemoveValues

```
// One loop over gBitGrpLists, combining pairs of values that differ in exactly one bit. If
// for a given value such a compatible value is found (referred to as 'buddy' in many
// places in the code), they are combined. This is done by 'throwing away' the value
// that has a '1' in the bit position that differs and then clearing the same bit in the
// remaining value's mask.
//
long RemoveValues( void )
{
  long            valuesRemoved = 0;
  long            curIdx = 0;
  BitGrpEntryPtr  beforeEntry;
  BitGrpEntryPtr  curEntry;
  BitGrpEntryPtr  *curList   = gBitGrpLists;
  BitGrpEntryPtr  *pastList =
    &gBitGrpLists[ gBitGrpMapper.numGrpLists ];

  if (gBitGrpMapper.chunkSize == 1) {

    while (curList < pastList) {
      if (*curList) {
        beforeEntry = (BitGrpEntryPtr)curList;
        curEntry = *curList;
        RemoveLoop1Bit( curIdx, beforeEntry, curEntry );
        valuesRemoved++;
      }
      curList++;
      curIdx++;
    }

  } else {

    while (curList < pastList) {
      if (*curList) {
        beforeEntry = (BitGrpEntryPtr)curList;
        curEntry = *curList;
        do {
          RemoveLoop( curIdx, beforeEntry, curEntry );
          valuesRemoved++;
          if (beforeEntry->next == curEntry) {
            beforeEntry = curEntry;
          }
          curEntry = curEntry->next;
        } while (curEntry);
      }
      curList++;
      curIdx++;
    }
  }

  return valuesRemoved;
}
```

RemoveLoop

```
// RemoveLoop deals with chunkSize == 2, 4 and 8. This Function loops over
// curEntry's buddy lists (lists that may contain values that, compared with the value in
// curEntry, differ in exactly one bit).
//
void  RemoveLoop( long            curIdx,
                  BitGrpEntryPtr  beforeEntry,
                  BitGrpEntryPtr  curEntry )
{
  short           curChunk = gBitGrpMapper.chunkCount;
  ulng            mask = gBitGrpMapper.firstMask;
  ulng            matchBit = gBitGrpMapper.firstMatchBit;
  long            magIdx = curIdx;
  long            magStep;
  ulng            scanVal;
  BitGrpEntryPtr  *buddyList;

  while (curChunk) {

    scanVal = curEntry->value & ~mask;
    magStep = gBitGrpMapper.lbTable[curChunk];
    if (magIdx < gBitGrpMapper.ubTable[curChunk]) {

      buddyList = &gBitGrpLists[ curIdx + magStep ];
      if (*buddyList) {
        if (ScanAndRemove( curEntry,
            (BitGrpEntryPtr)buddyList, ~mask, matchBit )) {

          return; // Found a 'buddy'
        }
      }
    }

    if (magIdx >= magStep) {
      buddyList = &gBitGrpLists[ curIdx - magStep ];
      if (*buddyList) {
        if (ScanAndKeep( curEntry,
          (BitGrpEntryPtr)buddyList, ~mask, matchBit )) {

                    // remove curEntry from list
          beforeEntry->next = curEntry->next;
          return;  // Found a 'buddy'
        }
      }

      do {
        magIdx -= magStep;
      } while (magIdx >= magStep);
    }

    mask >>= gBitGrpMapper.chunkSize;
    matchBit >>= gBitGrpMapper.chunkSize;
    curChunk--;
  }

  // No match found -> add to output and remove from list
  AddToOutput( curEntry );
  beforeEntry->next = curEntry->next;

}
```

RemoveLoop1Bit

```
// The 1 bit only version of RemoveLoop()
//
void  RemoveLoop1Bit ( ulng             curIdx,
                       BitGrpEntryPtr   beforeEntry,
                       BitGrpEntryPtr   curEntry )
{
  ulng            matchBit = gBitGrpMapper.firstMatchBit;
  BitGrpEntryPtr  *buddyHead;

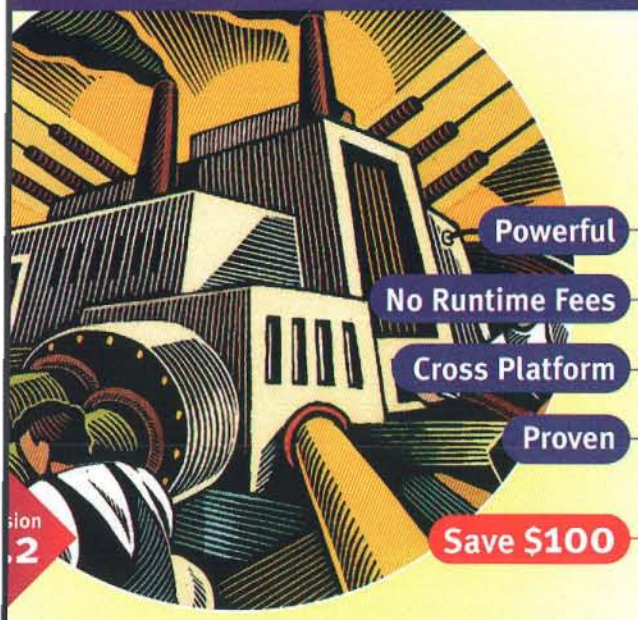
  while (matchBit) {

    if (curIdx & matchBit) {

      if (buddyHead = &gBitGrpLists[ curIdx & ~matchBit ]) {
        if ((*buddyHead)->mask == curEntry->mask) {
          (*buddyHead)->mask &= ~matchBit;
          beforeEntry->next = NULL;
          return;
```

Power Too Abundant to Meter!

neo•logic
Powering Development of Object-Oriented Applications



NeoAccess is the most powerful object-oriented database engine available. It displays electrifying performance—up to ten times that of its competitors. Behind its elegant programming interface is a fully optimized relational query engine tuned for short access times and an object cache with automatic garbage collection for nearly instantaneous access to previously used objects. All in a memory footprint as small as 150K.

Get the power of NeoAccess, and avoid the expense and administrative hassle of feeding the runtime fees meter. You pay one affordable price no matter how many copies of your application you sell or use.

Others may promise cross-platform development tools—NeoLogic delivers. NeoAccess is a set of C++ classes designed for use with popular compilers and application frameworks on Windows®, Macintosh®, and Unix™ platforms. Source code is included so it can even be used with custom frameworks.

Thousands of developers, including America Online® and Claris®, have already found that NeoAccess enabled them to build fast, powerful internet applications in record time. That's why there are more copies of NeoAccess based applications on end-user machines than any other object database backend. Tap the power!

neoAccess®
Cross-Platform Object Database Engine

Order Now!

M–F 9AM to 5PM Pacific: 1-800-919-6353
For Information and Customer Service: 1-510-524-5897

Order Directly from Our Web Site!
Download the Architectural Overview
www.neologic.com

NeoLogic Systems, Inc.
1450 Fourth St., Suite 12, Berkeley, CA 94710
v. 510.524.5897 f. 510.524.4501 neologic@neologic.com


```
        }
      }

    } else {

      if (buddyHead = &gBitGrpLists[ curIdx | matchBit ]) {
        if ((*buddyHead)->mask == curEntry->mask) {
          curEntry->mask &= ~matchBit;
          *buddyHead = NULL;
          return;
        }
      }
    }


    matchBit >>= 1;
  }

  AddToOutput( curEntry );

  beforeEntry->next = NULL;
}
```

ScanAndKeep

```
// For a given value, scans through a group list and searches for a buddy for that value.
// If a buddy is found, compareEntry will be removed by RemoveValues().
//
long  ScanAndKeep(    BitGrpEntryPtr  compareEntry,
                      BitGrpEntryPtr  beforeEntry,
                      ulng            mask,
                      ulng            matchBit )
{
  BitGrpEntryPtr  thisEntry = beforeEntry->next;
  ulng            scanValue = compareEntry->value & mask;

  while (thisEntry) {
    if ((thisEntry->value & mask) == scanValue) {
      if (compareEntry->mask == thisEntry->mask) {
        if (Match( compareEntry, thisEntry, matchBit)) {

          thisEntry->mask &= ~(compareEntry->value ^
            thisEntry->value);
          return 1;
        }
      }
    }
    beforeEntry = thisEntry;
    thisEntry = thisEntry->next;
  }

  return 0;
}
```

ScanAndRemove

```
// Same as ScanAndKeep() except that if a buddy is found, it is removed after the
// compareEntry's mask has been updated (see RemoveValues()).
//
long  ScanAndRemove(  BitGrpEntryPtr  compareEntry,
                      BitGrpEntryPtr  beforeEntry,
                      ulng            mask,
                      ulng            matchBit )
{
  BitGrpEntryPtr  thisEntry = beforeEntry->next;
  ulng            scanValue = compareEntry->value & mask;

  while (thisEntry) {
    if ((thisEntry->value & mask) == scanValue) {
      if (compareEntry->mask == thisEntry->mask) {
```

```
        if (Match( compareEntry, thisEntry, matchBit)) {

          compareEntry->mask &= ~(compareEntry->value ^
            thisEntry->value);
          beforeEntry->next = thisEntry->next;
          return 1;
        }
      }
    }
    beforeEntry = thisEntry;
    thisEntry = thisEntry->next;
  }

  return 0;
}
```

Match

```
// The two values compareEntry->value and thisEntry->value can be combined if they
// differ in exactly one bit. In those cases, refVal, below, will have exactly one bit set.
// The rest of the code tests to see if that is so. I have the feeling that this function's
// efficiency could be improved.
//
inline long  Match(   BitGrpEntryPtr  compareEntry,
                      BitGrpEntryPtr  thisEntry,
                      ulng            matchBit )
{
  ulng  refVal = compareEntry->value ^ thisEntry->value;
  ulng  matchCount = 0;

  switch (gBitGrpMapper.chunkSize) {
    case 2:
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      return 0;
    case 4:
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      return 0;
    case 8:
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      matchBit >>= 1;
      if (! (refVal ^ matchBit)) return 1;
      return 0;
  }
  return 0;
}
```

CleanUp

```
// I first developped this solution using the Symantec environment where I used the
// C++ new and delete functions for memory management. After moving to
// CodeWarrior, however, I had to use the Mac Toolbox function NewPtr to allocate
// memory in Init() (and DisposPtr to dispose of it here) because Codewarrior's
// implementation of new didn't seem to reliably return NULL in cases a memory
// request could not be satisfied.
//
long    CleanUp( void )
{
  if (gBitGrpLists) DisposPtr( (char*)gBitGrpLists );

  if (gRulesLeft > 0) {
    gCurRule->value = 0;
    gCurRule->mask = 0;
    if (gAllow == kAllow) {
      gCurRule->allow = kDeny;
    } else {
      gCurRule->allow = kAllow;
    }
    gRulesLeft--;

    return gMaxRules - gRulesLeft;
  } else {
    return -1;
  }

}
```

RouterRules

```
// Main entry point
//
long RouterRules( long  allowedValues[],
                  long  numAllowedValues,
                  long  numBits,
                  Rule  rulesArray[],
                  long  maxRules )
{
  gAllowedValues = allowedValues;
  gNumAllowedValues = numAllowedValues;
  gNumBits = numBits;
  gCurRule = rulesArray;
  gMaxRules = maxRules;
  gRulesLeft = maxRules;
  gStartMask = 0xffffffff >> (32-gNumBits);
  gAllow = kAllow;

  if (maxRules <= 0) return -1;

  if (numAllowedValues <= 0) {
    gCurRule->mask = 0;
    gCurRule->value = 0;
    gCurRule->allow = kDeny;
    return 1;
  }
  if (numAllowedValues == (1L << numBits)) {
    gCurRule->mask = 0;
    gCurRule->value = 0;
    gCurRule->allow = kAllow;
    return 1;
  }

  #if ALTER_INPUT_VALUES == 1
  if (numBits > 5 && numBits < 32 && numAllowedValues >
    ( (1L<<(numBits-1)) + (1L<<(numBits-5)) )) {
    MakeComplement();
  }
  #endif

  if (Init() == kErr) return -1;
  Process();
  return CleanUp();
}
```

Visit MacTech Magazine's Web site!
http://www.mactech.com

BITGRPMAPPER.CP

```
// ----------------------------------------
// Implementation of class BitGrpMapper - a class that does most of the base 2, 3, 5 and
// 9 arithmetic
// ----------------------------------------

#define WRITE_LOOKUP_TABLES     0

#if WRITE_LOOKUP_TABLES == 1
# include <stdlib.h>
# include <fstream.h>
#endif

#include "BitGrpMapper.h"

long BitGrpMapper::ubTable[33] = { 0 };
long BitGrpMapper::lbTable[33] = { 0 };

void BitGrpMapper::Init(  long        chunkSz,
                          long        numBits )
{
  switch (chunkSz) {
    case 1:
      lookupTable = NULL;
      firstMask = 0x01;
      firstMatchBit = 0x01;
      break;
    case 2:
      lookupTable = lookupTable2;
      firstMask = 0x03;
      firstMatchBit = 0x02;
      break;
    case 4:
      lookupTable = lookupTable4;
      firstMask = 0x0f;
      firstMatchBit = 0x08;
      break;
    case 8:
      lookupTable = lookupTable8;
      firstMask = 0xff;
      firstMatchBit = 0x80;
      break;
  }

// Calculate the upper bound and lower bound lookup tables used in RemoveLoop()
// of RouterRules.cp
  chunkSize = chunkSz;
  chunkCount = numBits / chunkSz;
  long excessBits = numBits % chunkSz;
  if (excessBits) chunkCount++;
  firstMask <<= ((chunkCount - 1) * chunkSz);
  firstMatchBit <<= ((chunkCount - 1) * chunkSz);

  ulng base = chunkSz + 1;
  ulng curBase = 1;
  long i;

  for (i=1; i<chunkCount; i++) {
    lbTable[i] = curBase;
    ubTable[i] = chunkSz * curBase;
    curBase *= base;
  }
  lbTable[i] = curBase;
  if (excessBits) ubTable[i] = excessBits * curBase;
  else            ubTable[i] = chunkSz * curBase;
  numGrpLists = ubTable[i] + curBase;

#if WRITE_LOOKUP_TABLES == 1
  {
    IndexEntry  tmpTable[256];
    ofstream    file;

    file.open( "BitGrpMapperTables.cp" );
    file << "#include \"BitGrpMapper.h\"" << endl << endl;
    CalcLookupTable( tmpTable, 2 );
    WriteLookupTable( file, tmpTable, "lookupTable2" );
    CalcLookupTable( tmpTable, 4 );
    WriteLookupTable( file, tmpTable, "lookupTable4" );
    CalcLookupTable( tmpTable, 8 );
    WriteLookupTable( file, tmpTable, "lookupTable8" );
```


TestTrack

Bug Tracking the Macintosh Way

1-2-3 Start Tracking
TestTrack is ready to use right out of the box—simply install, add a few users, and start tracking. It's that easy.

TestTrack is more than an easy-to-use bug tracking program—it's a powerful quality control tool for busy software development teams:

Automate TestTrack automates the tedious and error-prone process of reporting and tracking bugs by hand. It also eliminates the need to create a custom solution using general purpose database tools such as 4D™ or FileMaker Pro™.

Communicate TestTrack links engineers, testers, managers, even tech writers together so no one falls out of the loop. Team members are notified automatically when defects are assigned to them, guaranteeing communication and ensuring efficient work flow.

Stay up to Date TestTrack lets any authorized user look up the current state of any defect at any time.

Analyze TestTrack makes reporting easy—point, click, print and read. Customize reports to list what you want to see.

For a limited time, you can buy TestTrack for the introductory price of $99! Volume pricing, and site licenses are also available.

To order call 513-683-6456

Seapine Software, Inc. 1066 Seapine Ct. Maineville, OH 45039

sales@seapine.com
http://www.seapine.com

Seapine Software



Deciding on your development tools should be this easy too...



Details at...
www.SmalltalkAgents.com
Integrated Development Environment

Quasar Knowledge Systems, Inc.
(415) 728-5333 fax: (415) 728-1757 info@qks.com

```
    file.close();
  }
#endif
}

#if WRITE_LOOKUP_TABLES == 1

void BitGrpMapper::CalcLookupTable( IndexEntry  lookupTbl[],
                                    long        chunkSz )
{
  long  lookupByte;               // 0 .. 255
  long  byteInLong;               // 3 .. 0, 0 for the most sig. byte
  long  chunkCount = 8 / chunkSz; // 8, 4, 2 or 1
  long  curChunk;                 // 0 .. chunkCount-1
  long  bitCounts[8];             // bitCounts[0]: right most chunk
  long  curBit;                   //  bit 0 .. bit 7 (right to left)
  long  bitIndex;                 // 0 .. chunkSz-1
  long  base = chunkSz + 1;       // base 2, 3, 5, or 9
  long  curBase;                  // a power of base

  for (lookupByte=0; lookupByte<256; lookupByte++) {

    curBit = 1;
    for (curChunk=0; curChunk<chunkCount; curChunk++) {
      bitCounts[curChunk] = 0;
      for (bitIndex=0; bitIndex<chunkSz; bitIndex++) {
        if (lookupByte & curBit) bitCounts[curChunk]++;
        curBit <<= 1;
      }
    }

    curBase = 1;
    for (byteInLong=3; byteInLong>=0; byteInLong--) {
      lookupTbl[lookupByte].index[byteInLong] = 0;
      for (curChunk=0; curChunk<chunkCount; curChunk++) {
        lookupTbl[lookupByte].index[byteInLong] +=
          curBase * bitCounts[curChunk];
        curBase *= base;
      }
    }
  }
}

void BitGrpMapper::WriteLookupTable( ofstream    &file,
                                     IndexEntry table[],
                                     char*      tableName )
{
  long  entryCount;
  long  indexCount;

  file << "IndexEntry BitGrpMapper::"
    << tableName << "[256] = {" << endl;

  for (entryCount=0; entryCount<256; entryCount++) {
    file << "  { ";
    for (indexCount=0; indexCount<4; indexCount++) {
  // Printing as pointer to long writes value as a four byte hex number in the
  // Symantec environ. Not so with my brand new CodeWarrior (at least using the
  // default project settings). Would need to be fixed if the lookup tables had to be
  // rebuilt.
      file << (long*) table[entryCount].index[indexCount];
      file << ((indexCount == 3) ? " }" : ", ");
    }
    file << ((entryCount == 255) ? ' ' : ',' ) << endl;
  }

  file << "};" << endl << endl;
}
#endif
```


Want to know what products are available for MacOS development? Check out Developer Depot™ <http://www.devdepot.com>

BitGrpMapper.h

```
// ---------------------------------------
// class BitGrpMapper
// ---------------------------------------

#ifndef NULL
const void * const NULL = 0;
#endif

typedef unsigned long ulng;

struct IndexEntry {
..long..index[4];
};

class BitGrpMapper {..
public:
  static long  ..ubTable[33];
  static long  ..lbTable[33];
  long     chunkSize;    // 1, 2, 4 or 8 bits per chunk
  long     chunkCount;   // # of chunks/digits in group index
  ulng     firstMask;    // used to reset curMask
  ulng     firstMatchBit;
  long     numGrpLists;

  void.. ..Init( long..chunkSize,
                 long..numBits );

  inline long  LookUp( ulng      value );
private:
  static IndexEntry lookupTable2[];
  static IndexEntry lookupTable4[];
  static IndexEntry lookupTable8[];

  IndexEntry      *lookupTable;   // lookup group index

#if WRITE_LOOKUP_TABLES == 1
  void     CalcLookupTable( IndexEntry  lookupTbl[],
                            long chunkSz );
  void WriteLookupTable( ofstream &file,
                         IndexEntry       table[],
                         char     tableName );
#endif..
};

inline long BitGrpMapper::LookUp( ulng..value )
{
  long..index = lookupTable[value >> 24].index[0];

  index += lookupTable[(value >> 16) & 0xff].index[1];
  index += lookupTable[(value >> 8) & 0xff].index[2];
  index += lookupTable[value & 0xff].index[3];

  return index;
}
```

ProvidedCode.h

```
// ---------------------------------------
// Code copied from problem statement
// ---------------------------------------

enum {   kDeny = 0,
         kAllow = 1
};

#ifdef __cplusplus
extern "C" {
#endif
typedef struct Rule {
  long   mask;
  long   value;
  long   allow;
} Rule;
long RouterRules( long   allowedValues[],
                  long   numAllowedValues,
                  long   numBits,
                  Rule   rulesArray[],
                  long   maxRules );
#ifdef __cplusplus
}
#endif
```

BITGRPMAPPERTABL ES.CP

```
#include "BitGrpMapper.h"

IndexEntry BitGrpMapper::lookupTable2[256] = {
  { 0x00000000, 0x00000000, 0x00000000, 0x00000000 },
  { 0x00081BF1, 0x000019A1, 0x00000051, 0x00000001 },
  { 0x00081BF1, 0x000019A1, 0x00000051, 0x00000001 },
  { 0x001037E2, 0x00003342, 0x000000A2, 0x00000002 },
  { 0x001853D3, 0x00004CE3, 0x000000F3, 0x00000003 },
  { 0x00206FC4, 0x00006684, 0x00000144, 0x00000004 },
  { 0x00206FC4, 0x00006684, 0x00000144, 0x00000004 },
  { 0x00288BB5, 0x00008025, 0x00000195, 0x00000005 },
  { 0x001853D3, 0x00004CE3, 0x000000F3, 0x00000003 },
  { 0x00206FC4, 0x00006684, 0x00000144, 0x00000004 },
  { 0x00206FC4, 0x00006684, 0x00000144, 0x00000004 },
  { 0x00288BB5, 0x00008025, 0x00000195, 0x00000005 },
  { 0x0030A7A6, 0x000099C6, 0x000001E6, 0x00000006 },
  { 0x0038C397, 0x0000B367, 0x00000237, 0x00000007 },
  { 0x0038C397, 0x0000B367, 0x00000237, 0x00000007 },
  { 0x0040DF88, 0x0000CD08, 0x00000288, 0x00000008 },
  { 0x0048FB79, 0x0000E6A9, 0x000002D9, 0x00000009 },
  { 0x0051176A, 0x0001004A, 0x0000032A, 0x0000000A },
[ … and more - see online archive for complete file]
  { 0x02809F5F, 0x0007E8AF, 0x000018FF, 0x0000004F },
  { 0x0288BB50, 0x00080250, 0x00001950, 0x00000050 }
};

IndexEntry BitGrpMapper::lookupTable4[256] = {
  { 0x00000000, 0x00000000, 0x00000000, 0x00000000 },
  { 0x00003D09, 0x00000271, 0x00000019, 0x00000001 },
  { 0x00003D09, 0x00000271, 0x00000019, 0x00000001 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x00003D09, 0x00000271, 0x00000019, 0x00000001 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x0000B71B, 0x00000753, 0x0000004B, 0x00000003 },
  { 0x00003D09, 0x00000271, 0x00000019, 0x00000001 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x0000B71B, 0x00000753, 0x0000004B, 0x00000003 },
  { 0x00007A12, 0x000004E2, 0x00000032, 0x00000002 },
  { 0x0000B71B, 0x00000753, 0x0000004B, 0x00000003 },
  { 0x0000B71B, 0x00000753, 0x0000004B, 0x00000003 },
  { 0x0000F424, 0x000009C4, 0x00000064, 0x00000004 },
  { 0x0001312D, 0x00000C35, 0x0000007D, 0x00000005 },
[ … and more - see online archive for complete file]
  { 0x00057BCF, 0x00003827, 0x0000023F, 0x00000017 },
  { 0x00057BCF, 0x00003827, 0x0000023F, 0x00000017 },
  { 0x0005B8D8, 0x00003A98, 0x00000258, 0x00000018 }
};
IndexEntry BitGrpMapper::lookupTable8[256] = {
  { 0x00000000, 0x00000000, 0x00000000, 0x00000000 },
  { 0x000002D9, 0x00000051, 0x00000009, 0x00000001 },
  { 0x000002D9, 0x00000051, 0x00000009, 0x00000001 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x000002D9, 0x00000051, 0x00000009, 0x00000001 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x0000088B, 0x000000F3, 0x0000001B, 0x00000003 },
  { 0x000002D9, 0x00000051, 0x00000009, 0x00000001 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x0000088B, 0x000000F3, 0x0000001B, 0x00000003 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
  { 0x0000088B, 0x000000F3, 0x0000001B, 0x00000003 },
  { 0x0000088B, 0x000000F3, 0x0000001B, 0x00000003 },
  { 0x00000B64, 0x00000144, 0x00000024, 0x00000004 },
  { 0x000002D9, 0x00000051, 0x00000009, 0x00000001 },
  { 0x000005B2, 0x000000A2, 0x00000012, 0x00000002 },
[ … and more - see online archive for complete file]
  { 0x000013EF, 0x00000237, 0x0000003F, 0x00000007 },
  { 0x000013EF, 0x00000237, 0x0000003F, 0x00000007 },
  { 0x000016C8, 0x00000288, 0x00000048, 0x00000008 }
};
```

MT

WINREZ

Don't rewrite your apps. Port them!

WinRez is a complete emulation of all Resource Manager and Memory Manager routines.

Read/Write/Create your rez forks in your Windows apps.

String resources and rez fork chaining fully supported.

WWW.PARTIUM.COM

Want to share a tip with the community and get paid for it? Send it in to

<mailto:tips@mactech.com>

By Jamie Osborne, Apple Computer, Inc.

Shift into High Gear with Apple Game Sprockets

Turning your game idea into a reality, step by step

In the Beginning, There was a Cool Idea...

So, you want to write a cool game for the Macintosh. Well, you're in luck. This article will take you step-by-step through the whole process, beginning with an idea for a game, all the way to playing with your friends over a network. It will deal with loading and blitting sprites to the screen, getting input from any input device, no matter how snazzy, generating sounds that are correctly attenuated for distance and location, and doing all the work necessary to play your game over a network.

If that sounds like way too much to cover in a single MacTech article, you're right. At least, you would have been right if this article were written a year ago. Since the advent of Apple Game Sprockets, it is now possible to discuss everything you need to know (with a small amount of strategic glossing-over) in the space of a single article. I won't be able to delve deeply into each of the Sprockets, but that's OK because there's an on-line reference manual for that. I'll also gloss over some of the details about how to do things, such as how to load or store resources, or what goes into the application shell. However, the entire source for the game I will develop here is available with this issue's code disk, as well as on the Internet.

Now, what game am I going to do? A snazzy 3D blood bath like Marathon? No. A multi-player flight simulator that can have hundreds of people playing together over the Internet, a la Warbirds? No.

I am going to start off with something that has proven game play, yet is straightforward enough that I can actually get it done. I am going to do a game called "SprocketInvaders." Why? For two very good reasons. First, it's doable. For every game that you've seen on the store shelves or in the shareware section of INFO-MAC, there are a dozen other games that were started with the best intentions but never completed. Many of us have had good ideas for games. Some of us have even started games ourselves. However, very few of us have ever finished writing a computer game. Therefore, I am going to write something that I know I can do in a reasonably small period of time. The second reason is that despite (or perhaps because of) the fact that it is a very simple game, it lends itself very well to showing off the different areas I am concentrating on – rendering, sound, input, and networking.

Step One — Design

Designing SprocketInvaders was greatly aided by the fact that the "Invaders" game archetype is firmly established. Of course, you must make your game sufficiently unique so that you don't violate anyone's copyright. My design was pretty easy. I knew that I wanted aliens that moved in the invader style. I wanted a ship that would be able to move left, move right, and fire. I wanted the waves to get progressively harder by speeding up the rate at which the aliens moved and fired.

When you're starting a new game, it's easy to get bogged down in over-design. With SprocketInvaders that was not a problem, but with something more complex, perhaps a game where each level introduced new aliens or weapons, you might find yourself spending a whole bunch of time designing and not much time implementing. Of course, designing before implementing a game, or any software for that matter, is better

Jamie Osborne is the manager of Apple's Game Technology Group, which created the Apple Game Sprockets. He enjoys the fact that he can play games and get paid for it.

than implementing first and then designing as an afterthought. Again, the best way is to set attainable goals, achieve those goals, and build on your success. Design your game with one level, get it all running, then use your success to work on more sophisticated features or levels.

Step Two — The Shell

The best way to begin any new game is to start with a good application shell. This is true on any platform. Over the years, thousands of people have written thousands of Macintosh applications, nearly all of which had the same design.

- Initialize the Toolbox and globals.
- Do the event loop.
- Quit.

There is much more complexity than I've outlined here — handling events, managing the menu bar, etc. I have included a Macintosh shell application that is designed specifically for writing games that use the Apple Game Sprockets. It is a CodeWarrior project, but you should be able to use the source code in any development environment. You might notice that the shell code in SprocketInvaders differs significantly from the Game Shell I am providing. This is because the shell was done after SprocketInvaders was written. Since I really want to talk about the game itself, I will leave it to you to check out the shell.

Step Three — Artwork

Artwork is not really a stage, but more of an ongoing process. The better the artwork, the more impressive your game will look to the end user. The artwork for SprocketInvaders was done before the game was even started, since the main programmer's hobby is artwork, he enjoyed doing it up front. He is the exception. Most programmers cringe when they think about doing art for their game, and they have good reason — most people are not good artists.

The best recommendation I have about doing your art is that you find someone who has at least *some* talent for it. If that means you, fantastic! If not, you should call up friends and relatives looking for someone who has a knack for it. The artwork in SprocketInvaders will not likely wind up in a museum of computer art, but it's not too shabby. If you can't manage to make art as good as the art in SprocketInvaders, find someone who can. As a last resort, cajole them into helping you with promises of fame and fortune. Many amateur artists are willing to do it just to have their name in the About box. A small price to pay for a better game.

If you plan on selling your game commercially you *must* have your art done by someone who is a professional, or at least does professional-quality art. However, if you hope to have some company publish your game, you might be able to get away with doing your own art for the first versions and letting the publisher redo the artwork. Doing that, of course, will cost you some of the profit.

On the technical side of things, the sprites were drawn in a graphics program, then stored in 'SPRT' resources that include the height and width of the sprite, as well as some information about the different versions used to animate the sprites.

Listing 1: Sprite.h

```
typedef struct Frame
{
  UInt16 height, width;
  SInt16 hotX, hotY;
  UInt8  *image;
} Frame, *FramePtr;

typedef struct Sprite
{
  UInt16   numFrames;
  FramePtr *frames;
} Sprite, *SpritePtr;
```

The Sprite structure is simply a pointer to an array of Frames, with a count. The Frame structure is a little more interesting. It contains the pointer to the actual pixmap (the image), as well as information about the height and width of the sprite. Keep in mind that the pixmap is bit-depth dependent. In the case of SprocketInvaders, all of the artwork is 8-bit, and the user's monitor must be in 8-bit mode when the game runs, or else evil things will happen. Luckily, DrawSprocket takes care of setting the user's monitor to the exact bit depth and resolution you specify, so your game will always be guaranteed to look right. If you want your game to be playable in several bit depths, you need to create artwork for all of the bit-depths that you want to support.

You should be able to use the sprite libraries in SprocketInvaders "right out of the box" for your game. There are some dependencies on other parts of the code, such as the blitters, so you might need to make modifications if you intend to deviate very much from SprocketInvaders' sprite/graphic routines. There are also some very good sprite libraries available on the Internet.

Step Four — Initialization

Initialization is one of those things that should be a snap, right? Call a couple of Toolbox init functions, load in some data, and you're done. Well, the answer is "yes and no." Once you have a good game shell and you're familiar with all the different services you will be using, such as the Game Sprockets, initialization *is* easy. However, the first time can be pretty tricky, and there are many snares you can set for yourself by incorrectly initializing your game. These snares can trap you later on in the form of bizarre crashes or strange behavior. That is why it is essential to get your initialization right. It's the price you pay up front for the convenience you get later on.

Listing 2: main.c

main Snippet

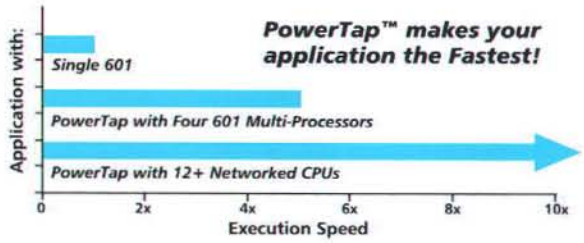
Main is where we do all our initialization.

```
ToolboxInit();

// Make sure that the DrawSprocket and InputSprocket libraries are present
if (DSpStartup == nil)
  FatalError("This game requires DrawSprocket.");
```

No Pain...All Gain
with
PowerTap™

Add multiprocessing and cross-network processing speed to your applications today!



PowerTap is a robust, easy-to-use software library used by Macintosh software developers to make their applications lightning-fast! It greatly improves performance on processor intensive applications by tapping the power of any Macintosh processor on a network.

Without having to learn about networking, communications, or task scheduling algorithms, developers can divide their application into tasks, and submit them to the **PowerTap** library, so that their user's can get results FAST! **PowerTap** applications can be written with CodeWarrior, Symantec, or MPW enviroments including LS Fortran as well as most other popular Macintosh compilers.

Download a free PowerTap demo at:
www.fortner.com/PowerTap_demo
Or request a free demo CD-ROM by calling:
1-800-252-6479

Special Introductory Price!
$299.00
Retail Price $395

FORTNER
RESEARCH LLC
100 Carpenter Dr. • Sterling, VA 20164
(800) 252-6479 • (703) 689-9593 FAX
www.fortner.com • info@fortner.com


```
if (ISpInit == nil)
  FatalError("This game requires InputSprocket.");

// Reseed the random number generator
qd.randSeed = TickCount();

startupModifiers = EventInit();
gNetSprocketPresent = InitNetworking();

MenuInit();

GraphicsInit(startupModifiers);
SoundHandlerInit(startupModifiers);
RedbookHandlerInit(startupModifiers);
```

The first thing you notice is that I check to make sure that DSpStartup and ISpInit are valid symbols, to ensure that the DrawSprocket and InputSprocket shared libraries are installed in the user's system. I could also do this by strong-linking the libraries when I build the SprocketInvaders application. However, doing it this way allows me to present the user with a slightly more friendly error message than the default one, "Couldn't start the application SprocketInvaders because an unexpected error occurred."

Input

In addition to doing some MacOS event-handling stuff, I initialize InputSprocket. At first glance, the code for InputSprocket initialization may seem utterly obtuse. However, it is not as tricky as it seems.

The myNeeds structure is simply a list of all the input elements wanted in order to play the game. For SprocketInvaders, I need a movement axis and fire button for each player, as well as a key to exit the game. Look at the first element in the needs list.

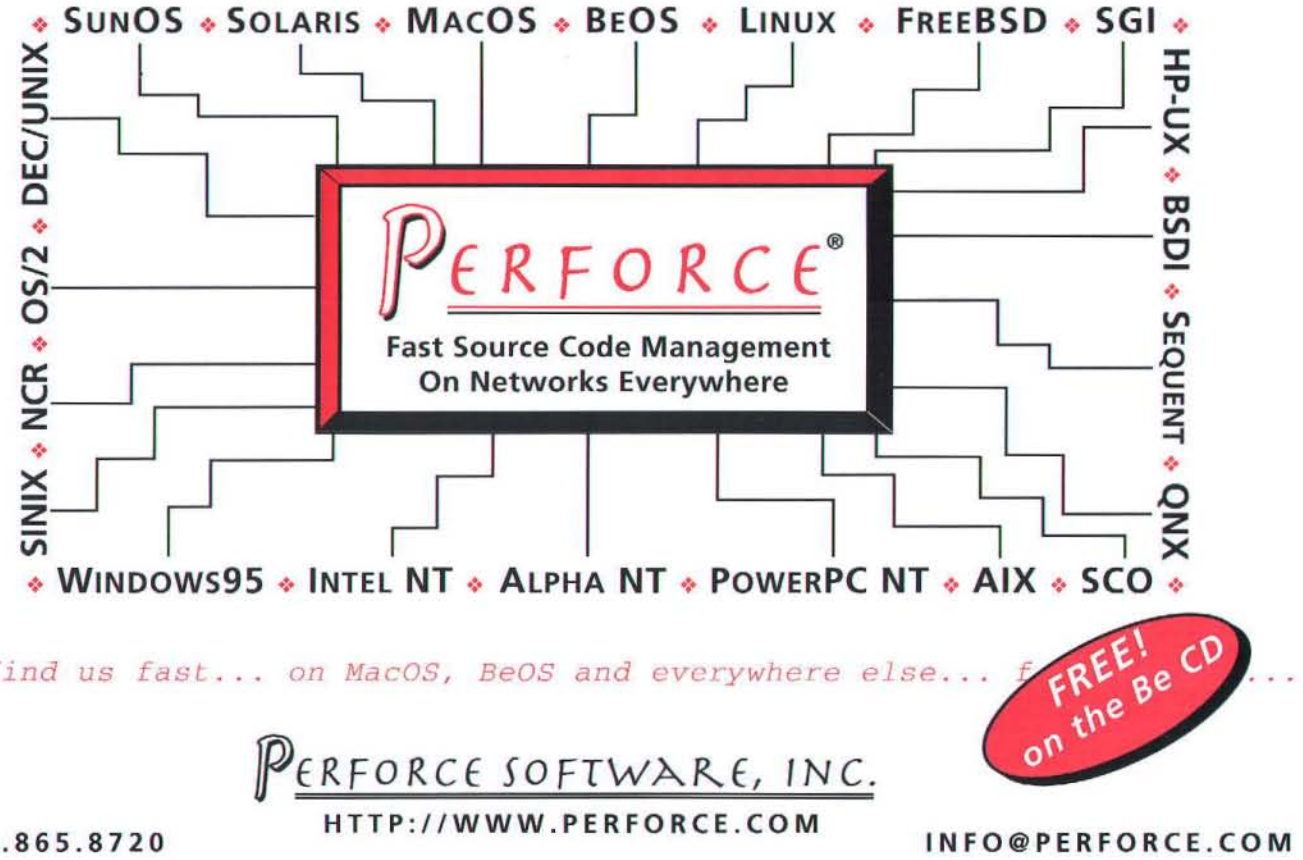
Listing 3: EventHandler.c

EventInit Snippet

```
{
  "\pGreen Movement",
  200,
  0,
  kISpElementKind_Axis,
  kISpElementLabel_XAxis,
  0,
  0,
  0,
  0
},
```

This is a pretty big structure to fill in, but most of the fields are used only for expert customization. For my needs, filling in a couple of the fields and leaving the rest set to 0 will do just fine. The ISpNeed structure requires:

- The name of the control for display in the configuration dialog.
- The icon suite resource id that InputSprocket should use in the configuration dialog. (You must create the icon and store it in your app's resource fork. You could just punt and specify 0 here, telling InputSprocket to use default icons, but that's lame.)
- A reserved field that should be set to 0.
- The kind of information wanted, such as an axis, button, or direction pad. In this case, I want a true analog axis.



- A label telling InputSprocket how you intend to interpret the element — in this case, the X (horizontal) axis. InputSprocket uses this information to provide default mappings from an input device to the input needs. For example, a joystick's horizontal axis will be mapped to the X axis need by default.
- Flags and reserved fields you don't need to worry about.

I set up the rest of the needs structure the same way, then call

```
theError = ISpElement_NewVirtualFromNeeds(numInputs, myNeeds,
 gInputElements, 0);
```

This turns the list of needs into an element list that I then pass to the ISpInit function. Again, this might seem like excessive work to set up input, but the reason I have to do so much here is that InputSprocket is extremely flexible. It needs to be able to support input devices that didn't even exist when the game was written. By using the InputSprocket API, the game will work with all devices that have an InputSprocket driver, present and future. Just think — your game could be the first one played with the Direct Neural Implant Controller in the year 2001!

Networking

Going back to main, I've handled InputSprocket initialization. Next up is initializing the network code.

Listing 4: NetSprocketSupport.c

InitNetworking Snippet

```
if (! NetSprocketPresent())
  return (false);

// Initialize NetSprocket
status = NSpInitialize(kStandardMessageSize, kBufferSize,
        kQElements, kGameID, kTimeout);
FAIL_OSSTATUS(status, "NSpInitialize failed");
```

Unlike DrawSprocket and InputSprocket, I don't *need* NetSprocket to be installed in order to play the game. If the user does have NetSprocket, I enable the game to be played over the network. If the user doesn't have it, I simply disable the menu item that allows the user to play over the network. They can still play with one or two players on a single machine. When linking the game, I import NetSprocketLib as a 'weak' link. Doing this allows the game to run even if NetSprocket isn't present. The key is to make sure that you do not try to call any of the library's functions if not present at runtime. I do this in SprocketInvaders by checking if the NSpInitialize function is valid (non-null) and setting the gNetSprocketPresent global flag accordingly.

Once I have determined that NetSprocket is present, I initialize it by calling NSpInitialize. The values you pass to NSpInitialize help NetSprocket optimize the way it handles your messages. You give it the 'standard' message size, which in the case of SprocketInvaders is the size of the player input message.

The next two parameters tell NetSprocket how much buffer space to pre-allocate. You can set both of these to 0 to have NetSprocket use default values. The game ID you pass should be your application's creator type. NetSprocket uses this as a unique identifier for your game, allowing it to pick a default TCP/IP port number or an AppleTalk NBP type that is unique for your game. The time-out parameter simply tells NetSprocket how long it should wait, in milliseconds, before giving up on certain network operations, such as making a connection.

Drawing

main does a few more initializations, then calls GraphicsInit:

Listing 5: Graphics.c

GraphicsInit Snippet

Initializes our color table, gets our buffers ready, and fades the screen.

```
DSpStartup();

// CLUT 8 is the standard 8-bit system color table and is always present
gGameCLUT = GetCTable(8);
DetachResource((Handle) gGameCLUT);

// Define and create our drawing context
GraphicsSelectContext(modifiers);

// Add our various colors to the color table
GraphicsMergeCLUT(128, kFirstIconColor, kLastIconColor);
GraphicsMergeCLUT(kPICTGasCloud, kFirstBGColor,
    kLastBGColor);
GraphicsMergeCLUT(kCLUTSprites, kFirstSpriteColor,
    kLastSpriteColor);

// Create our underlay and load a picture into it
GraphicsCreateUnderlay();
GraphicsLoadBackground(kPICTGasCloud);

// Swap the buffers once to get the underlay image on the screen
DSpContext_GetBackBuffer(gDisplayContext,
kDSpBufferKind_Normal, &backBuffer);
DSpContext_InvalBackBufferRect(gDisplayContext,
    &backBuffer->portRect);
GraphicsUpdateScreen();

// Fade up from black and then put the context in the paused state
// so that the user has access to the menu bar.
DSpContext_FadeGammaIn(nil, nil);
GraphicsPaused();
```

A color lookup table, or CLUT, is simply a table of colors that the video hardware uses when in 8-bit video mode. Technically, a CLUT is a MacOS resource that describes a palette. Since one cannot express a complete range of colors with only 8 bits, computers use indexed colors that simply create a palette of colors, much like a painter's palette, that the computer uses to draw everything. When using indexed colors, every pixel on the screen must be one of the colors in the palette. More sophisticated games use custom color tables designed to make the art in the game look very cool. If a game has no need to draw anything in yellow, for instance, yellow's spot in the palette can be used for another color — perhaps another shade of pinky-russet. An even more sophisticated use of CLUTs is to animate the color table, rapidly changing the contents of each entry in the table. This technique is used in Marathon and many other high-end games to produce stunning lighting effects.

However, SprocketInvaders really doesn't need to be that sophisticated. In fact, if it weren't for the neat background image used, I probably could have gotten away with using the standard system palette. I wanted everything to look just right though, so I took three different CLUTs — for the backdrop, the icons, and the sprites, and merged them all into one CLUT that DrawSprocket uses for the front buffer (the one that is actually displayed on the screen.) Once I've created the custom CLUT, I load in the backdrop PICT as an underlay, which is simply a buffer that contains the image that should be drawn under everything else.

The astute reader might note that I have glossed over the GraphicsSelectContext function. The reason is that the code is pretty self-explanatory. DrawSprocket quite handily allows you to specify exactly what kind of video environment you want your game to run in. If the user has a multi-sync monitor that uses 32-bit color in 1024x768 resolution, but your game wants to run in a 640x480 screen with 8-bit color, DrawSprocket can set the monitor up exactly as you want it, then set it back to exactly the way it was before the user ran your game. It may not sound like much work, and it's not hard for you to do using DrawSprocket, but I can tell you that using DrawSprocket to set your video mode saves you, the game developer, *numerous* headaches.

Once DSpFindBestContext returns a DrawSprocket context (an object from which I get the front and back buffers), I get the back buffer and invalidate it. Finally, I call DSpContext_SwapBuffers, causing DrawSprocket to move the back buffer to the front. Since the back buffer is invalid and I have set an underlay buffer, DrawSprocket copies the contents of the underlay to the front buffer, leaving a screen that contains the nice backdrop.

I should note that I also use DrawSprocket's gamma fade routines to fade out the screen before I do the mode switch (to 640x480x8), then fade back in. Gamma fades are a very nice way to hide the ugliness of switching monitor modes from the user. I highly recommend that you follow the same procedure when initializing your game.

Sound

The last service to initialize is sound. The sound initialization requires that I allocate new sound channels using the Sound Manager, as well as create new SoundSprocket objects, if SoundSprocket is installed.

Good sounds are a must for any game because they dramatically improve the enjoyability. For the amount of work done in coding, the payoff is huge. I use the Sound Manager in SprocketInvaders for all of my sounds. As with most games, I allocate a set of sound channels that I use for all the sounds played in the game. I decided to use the same number of channels as I had sounds, assigning each to a channel. If a sound is already playing in the channel when I start a new sound, the new sound simply preempts the old one.

This approach works just fine for SprocketInvaders, or for any game with a limited number of sounds. However, if your game uses lots of sounds, you should consider another approach to allocating sound channels, such as selecting an appropriate channel based on

how important it is for the user to hear the sound.

Once I have initialized the Sound Manager items, I initialize SoundSprocket, if it is present at runtime.

Listing 6: SoundHandler.c

SoundHandlerInit Snippet

Initializes the Sound Manager, then creates the necessary SoundSprocket objects to do 3D sound localization.

```
// Create the listener
theErr = SSpListener_New(&gListener);
if (theErr)
  FatalError("Could not create a sound sprocket
      listener.");

// Define our unit of distance measurement
theErr = SSpListener_SetMetersPerUnit(gListener, 0.05);
if (theErr)
  FatalError ("Could not set reference distance for
    listener.");

// Attach the sound localization component to each of the sound
// channels that we'll be playing through.
for (i = 0; i < soundNumSounds; i++)
{
  SoundComponentLink  myLink;

  // Create the source
  theErr = SSpSource_New(&gSource[i]);
  if (theErr)
    FatalError("Could not create a sound sprocket
      source.");

  // Install the filter
  myLink.description.componentType = kSoundEffectsType;
  myLink.description.componentSubType =
      kSSpLocalizationSubType;
  myLink.description.componentManufacturer = 0;
  myLink.description.componentFlags = 0;
  myLink.description.componentFlagsMask = 0;
  myLink.mixerID = nil;
  myLink.linkID = nil;

  theErr = SndSetInfo(gChannels[i],
            siPreMixerSoundComponent, &myLink);
   if (theErr)
    FatalError("Could not install the sound sprocket filter
      into the channel.");
}

gSoundSprocket = true;
SetListenerLocation();
```

All I need to do is allocate a new listener object (so that I'll know the relative locations of the sound generators), set the distance units (which are pretty arbitrary), and attach a localization component to each sound channel. The localization component is a filter placed into the Sound Manager that allows me to specify coordinates when I play the sound. The filter takes those coordinates and attenuates the sound accordingly.

Well, I said it wasn't easy. Initialization is hard work. I skipped some of the details in each area, and I didn't fully explain some of the services used — I will do that later. Remember, though, once you have initialized one game, you are well on your way to initializing many others.

Step Five — The Infinite Loop

I am referring to the Game Loop. GameLoop (in SprocketInvaders.c) is called by the application shell's main event loop (after it takes care of the business of handling MacOS events). This step is actually quite involved. It can be broken down into stages, each of which will require a good deal of explaining.

- Handle player input.
- Do collision detection.
- Handle game logic.
- Draw everything.

The event loop is a very important part of the game because it is the heartbeat that keeps the game alive. In addition to the stages listed above, there are some other sundry tasks the game loop routine takes care of — things like checking to see if the current wave of the game is over or the user wants to quit. Those pieces of code are straightforward MacOS programming. Let's look in more detail at the stages listed above.

Handle player input

In the earlier incarnations of the game I had an input implementation that used GetKeys. It took me under an hour to add InputSprocket support. Now, SprocketInvaders can be played with just about every conceivable input device available.

How to handle input depends on whether or not the game is being played over the network. When playing locally with either one or two players, I simply call GetGreenInput and GetRedInput. In a one-player game, the player is always the green player. In a two-player game, player two is the red player.

When playing over the network, I need a little bit of extra logic. The player who hosted the game (advertised it on the network for another to join) is always considered player one, the green player. The person who joined the game is player two, the red player. Therefore, when I do my input, the host player (gIAmHost is true) first reads the Green input from the local computer then sends it to the other player. The player who joined reads the Red input from the local computer and then sends it. Each then calls the Get*Player*Input function for the other player. The input for that other player will have come in via the network by the time the Get*Player*Input function returns.

Listing 7: SprocketInvaders.c

GameLoop Snippet

Get the input from both players.

```
if (gNetPlay)
{
  if (gIAmHost)
  {
    GetGreenInput();
    SendInputState(gGameKeys.greenLeft, gGameKeys.greenRight,
          gGameKeys.greenFire);
    GetRedInput();
  }
  else
  {
    GetRedInput();
    SendInputState(gGameKeys.redLeft, gGameKeys.redRight,
          gGameKeys.redFire);
    GetGreenInput();
  }
}
else
{
  GetGreenInput();
  if (gTwoPlayers)
    GetRedInput();
}
```

For now, I'll gloss over how the input is delivered via the network. Just take it as during net play calling the GetPlayerInput function for the remote player will get the necessary input over the network. I do, however, want to look more closely at what happens when getting the input for the local player.

I use InputSprocket for getting the player's input because it is by far the most reliable, expressive, and compatible input API that exists for the MacOS. Reliable because it has been extensively tested and, unlike other methods such as GetKeys, you'll never miss an input event. Expressive because you can tell InputSprocket your game's wildest needs. Need three analog axes, two buttons, and a toggle switch to play your game? No problem. InputSprocket maps your game's needs to the player's input devices. Compatible because you know it's going to work with most input devices available, including the keyboard and mouse, Thrustmaster FCS, WCS, and RCS (try playing SprocketInvaders with the rudders — it's wild!), all CH products, Gravis Game Pad, Gravis Firebird, etc. The list keeps growing.

Enough extolling of virtues. Let's look at the code for getting information from InputSprocket.

Listing 8: SprocketInvaders.c

GetGreenInput Snippet

Obtains the input state for the local player from InputSprocket.

```
UInt32 input;

// Check the movement axis
ISpElement_GetSimpleState(gInputElements[greenMovement],
      &input);

gGameKeys.greenLeft = false;
gGameKeys.greenRight = false;

if (input < 0x2FFFFFFF)
{
  gGameKeys.greenLeft = true;
}
else if (input > 0xBFFFFFFF)
{
  gGameKeys.greenRight = true;
}
else if (input < 0x5FFFFFFF)
{
  gGreenAccum += ((float) input - (float) 0x5FFFFFFF) /
      (float) 3FFFFFFF;
  if (gGreenAccum < -1)
  {
    gGameKeys.greenLeft = true;
    gGreenAccum += 1;
  }
 }
 else if (input > 0x9FFFFFFF)
 {
  gGreenAccum += ((float) input - (float) 0x9FFFFFFF) /
      (float)0x3FFFFFFF;
  if (gGreenAccum > 1)
  {
    gGameKeys.greenRight = true;
    gGreenAccum -= 1;
  }
 }

// Check the fire button
gGameKeys.greenFire =
GetAutoFireButton(gInputElements[greenFire]);
```

The game specifies what its "needs" are at initialization time — for SprocketInvaders, a left/right axis and a fire button. During the initialization, I expressed those needs to InputSprocket. Now, as I check the input for the player, I get the left/right movement information in the form of an unsigned value between 0 and 0xFFFFFFFF. 0 means the axis is all the way left. 0xFFFFFFFF means it is all the way right. The game has no idea how InputSprocket is getting this information from the user, and frankly, it doesn't care. All it knows is that InputSprocket and the user have agreed on how the user indicates left, right, and fire.

The first line of this function requests the axis information from InputSprocket. Remember that this information could be coming from a true analog axis, such as a joystick, or it could be coming from a digital source, such as the keyboard. In either case, you will get back an analog value. Because I would like to allow people with joysticks to finesse their movements, I do a little extra work than necessary to deal with a true analog device being moved only a portion of the distance between the center and the extreme limit. For instance, if the value I get back is between 0x2FFFFFFF and 0x5FFFFFFF (towards the left, but not quite all the way), then I use a global to accumulate left movement until I reach a threshold, at which point the player input for that iteration through the game loop becomes "move left."

Listing 9: SprocketInvaders.c

GetAutoFireButton Snippet

Determines whether the user is trying to fire.

```
// poll
error = ISpElement_GetSimpleState(inElement, &input);
if (! error && (input == kISpButtonDown))
  fire = true;
// but don't miss fast clicks or macros do
{
  ISpElementEvent event;
  error = ISpElement_GetNextEvent(inElement,
            sizeof(ISpElementEvent), &event, &wasEvent);

  if (! error && wasEvent && (event.data == kISpButtonDown))
  {
    fire = true;
    break;
  }
} while (wasEvent && !error);

// flush the queue
ISpElement_Flush(inElement);
```

Here I get the current state of the fire button (up or down), but I also check the InputSprocket event queue to see if there was a button down event. I check the event queue in case the user hit the fire button really quickly and then released. Checking the current state of the fire button would not tell me that this had happened, but getting the input events from InputSprocket will. This is a good example of how using InputSprocket is superior to other methods, such as using GetKeys.

Why do I check the button state and get the input events? The answer is simple. I want the user to be able to "auto-fire," so that holding down the fire button will cause the next shot to be fired as soon as the current one is done. While there are other ways to handle auto-fire, such as keeping the button state in a global variable, I felt that this implementation was the easiest method.

Do collision detection

The game maintains several lists, including players, enemies, and shots. Each time through the loop, I need to check to see if any of the enemies' shots have hit the players or any of the players' shots have hit the enemies. There are many subtle ways of doing collision detection. Nearly all of them come down to comparing one rectangle to another and determining if there is any overlap The more sophisticated methods reduce the number of comparisons done. SprocketInvaders uses the brute-force approach, which is just fine for its needs.

Listing 10: SprocketInvaders.c

CollideShotsToEnemies Snippet

Performs all collision detection for the players' shots and the invaders.

```
GameObjectPtr shot;
GameObjectPtr target;

if (! gEnemyList)
  return;

    // Iterate over the green player shots
if (gGreenPlayerShotList)
{
  for (shot = gGreenPlayerShotList; shot; shot = shot->next)

  {
    // Iterate over the enemies
    for (target = gEnemyList; target; target = target->next)
    {
      Rect u;
    // Union of shot and enemy screen rects

      SectRect(&shot->screenRect, &target->screenRect, &u);
      if (! EmptyRect(&u))
      {
        SoundHandlerPlay(soundEnemyHit, target->screenX,
              target->screenY);

        shot->action = PlayerShotDestroy;
        target->action = EnemyDestroy;
        AddPoints(target->screenX, target->screenY);
      }
    }
  }
}
```

For every shot the green player has created, I check each enemy in the enemy list to see if the shot's rectangle overlaps with the enemy's rectangle. If I determine there is an overlap, I set the shot's action proc to be PlayerShotDestroy, and then set the hapless enemy's action proc to be EnemyDestroy. Later in the game loop, I will execute these functions. I also play a sound effect and call AddPoints, which causes the little "100" sprite to appear. When there are two players, the CollideShotsToEnemies function repeats the same process for the red player's shots. After the game loop does collision detection for the players' shots, it checks the invaders' shots, using a similar process.

The part about setting the objects' action procs may sound confusing, but it really isn't. Each object in the game, as part of its data structure, includes a function pointer that, when non-null, is executed in the game logic phase of the game loop. It is an approach to dealing with the game actions that is object-oriented but does not require C++. Also, you might wonder why I have a shot list for each player, when each player can have only one shot at a time in the air. The reason is that I wrote the game to be extensible and flexible. In the design phase, I decided not to allow more than one shot. However, should I have changed my mind later on in the development process, it would have been trivial to change the game to allow more shots. By coding it in at the beginning, I remained flexible in the implementation. Flexibility is an extremely important part of game design and implementation, especially when radical changes (such as adding network support) come along.

Handle game logic

Let's face it. Invaders are dense. They move left and right in a completely predictable pattern and speed. As their numbers are reduced by the player's skill at shooting fish in a barrel, they get madder and madder, and thus move faster and shoot more often. They don't actually target the player. They don't follow the player at all. They are dumb.

Nonetheless, you need to have logic in your game, if the computer controls any entities in it. Even the simple task of making the Invaders behave in their single-minded manner took quite a bit of work. It is not as trivial as one would think.

Consider the way aliens speed up as their numbers decrease and the wave number increases. Of course, there are several potential ways of speeding up the aliens. You could assign each alien a velocity at the beginning of the wave and increase it as their numbers decrease. However, you might note that the aliens not only move faster and faster, but also shoot more often. You'd have to process all the aliens faster and faster as the game went by, perhaps requiring you to check the TickCount all the time. Clearly, this approach has a good deal of complexity.

I believe the solution I used is more elegant. To be honest, I was inspired by the way the original Space Invaders handled this problem. I set a global variable, gEnemyGas, that is increased at the beginning of each wave, but remains constant during the entire wave. This value is the total number of aliens processed each time through the game loop. For example, gEnemyGas is set to 2 for wave one. If you look closely, you will see that the aliens seem to move in pairs. That is because I process two of them each time through the game loop.

This may not seem very interesting until you consider what happens as you destroy the little buggers. At the beginning of the level, there are 5 rows of 11 aliens, totaling 55 invaders. Since I process only two of them each time through the loop, it takes 28 iterations through the game loop to move the entire swarm one step to the right (or left). During those 28 iterations, you are free to move around and shoot at your leisure. Put in more technical terms, your input is processed 28 times for each time any single invader's action proc is executed.

Now consider what happens as their numbers are reduced. If you destroy half of the invaders, you are left with 28 aliens. Since 2 aliens per iteration are processed, it now takes only 14 iterations of the game loop for a given alien to be processed twice. Relative to the number of times your input is processed, they are moving and firing more often. As the waves advance, and the value of gEnemyGas increases, the enemies will move

Features | Benefits

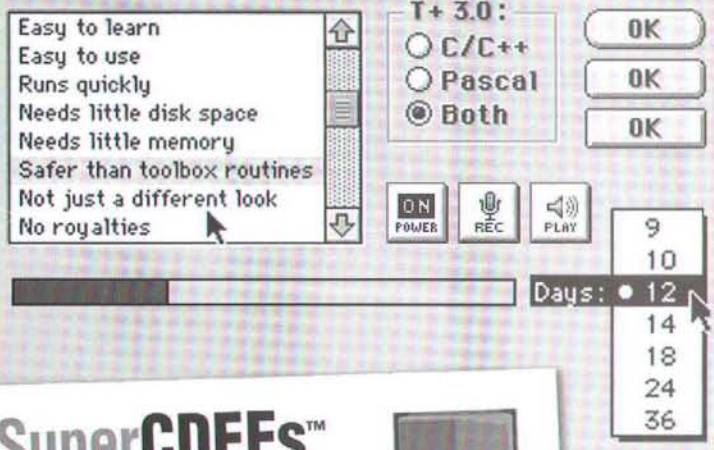
Create a *working* GUI element with just 1 line of code!

Tools Plus 3.1

Get extraordinary results using ordinary C, C++ or Pascal. Only 170 core routines (about 370 total) replace the need for thousands of Toolbox routines, and thousands of lines of code. Tools Plus has you covered with...

- **Windows** (all kinds, auto-positioning)
- **Cursors** (color, animation, auto-change)
- **Scroll bars** (speed control, live scrolling)
- **Picture buttons** (the best anywhere)
- **Makes CDEFs & LDEFs work automatically**
- **Panels** (3D, flat, many options)
- **Menus** (pull-down and hierarchical)
- **Editing fields** (w/scroll bars and filters)
- **Edit menu** (auto-editing with undo/redo)
- **Full color and multi-monitor support**
- **World-class custom buttons and sliders**
- **Tool bar**
- **Floating palettes**
- **Buttons** (all kinds)
- **List boxes**
- **Pop-up menus**
- **Clipboard**
- **Dynamic alerts** (no resources req'd)
- **Automatic event handling**
- **and much more!**

One line creates it. One line makes it work.



SuperCDEFs™

World-Class Controls for the Discerning Developer

For all Macintoshes and system versions

Full color and multi-monitor support

680x0, PowerMac and Fat/Safe format

The finishing touch for a complete 3D look



- ✔ 7 buttons, 9 tabs and 11 sliders (flat/3D text, flat/3D body, raised/inset title, soft/bold shadows, variable shapes) plus a thermometer
- ✔ Customizable check boxes plus undefined state
- ✔ Sliders options: tick marks (1 side/2 sides/none), scale (forward/reversed/none), snap or step to mouse, smart scaling





only $89*

*Free with purchase of Tools Plus Developer Kit

Water's Edge Software

2441 Lakeshore Road West
Box 70022
Oakville, Ontario
Canada L6L 6M9
Phone: (416) 219-5628
Fax: (905) 847-1638

e-mail: WaterEdgSW@aol.com
http://www.interlog.com/~wateredg

Tools Plus for
Symantec (THINK) C/C++ $149
THINK Pascal $149
THINK C/C++ & Pascal $199
CodeWarrior Bronze $199
CodeWarrior Gold $249

(We accept VISA and Amex. Add $10 for shipping.)
*Call for Academic pricing

Free Evaluation Kit: Available on AOL, CIS, Internet and at our web site

MW ★★★★ Macworld, Feb'96

and fire faster and faster. In fact, there comes a time in each wave when there is more gas than there are aliens. When that happens, the aliens have the upper hand, because they get processed multiple times for every time your input is handled. Eventually, the aliens will win. Resistance may be futile, but it can be alot of fun!

Let's look more closely at exactly how to "process" a single invader. During the initialization phase of each wave, I create and initialize each enemy invader. One part of that process is assigning the enemy's action proc to be EnemyAction. Until an external event occurs, such as being hit by the player's missile, this action proc is executed each time the alien is processed.

Listing 11: ObjectActions.c

EnemyAction Snippet

Defines the default behavior of an enemy.

```
 if (theObject->refCon > gEnemyLevel)
   return;
   theObject->refCon++;
gNumEnemiesProcessed++;

if (theObject->refCon & 1)
   theObject->frame = 1;
else
   theObject->frame = 0;

// Determine if this enemy needs to shoot
if (Game_Random((2 * kNumEnemies) - gNumEnemies) == 1)
   EnemyShoot(theObject);

// Adjust screen position depending on whether we're moving left or right
if (gEnemyTask == kEnemyMovingRight)
{
   theObject->screenX += gEnemyVelocity;
}
else if (gEnemyTask == kEnemyMovingLeft)
{
   theObject->screenX -= gEnemyVelocity;
}
else if (gEnemyTask == kEnemyDropping)
{
   theObject->screenY += kEnemyVDrop;
}

// Check for bottom bound
if (theObject->screenY > theObject->bounds.bottom)
{
   gNumRedPlayerLives = 0;
   gNumGreenPlayerLives = 0;
   return;
}

// Check left bounds
if (theObject->screenX < theObject->bounds.left &&
(gEnemyTask == EnemyMovingLeft))
{
   gEnemiesChangeDirection = kEnemyMovingRight;
   return;
}

// Check right bounds
if (theObject->screenX > theObject->bounds.right &&
(gEnemyTask == EnemyMovingRight))
{
   gEnemiesChangeDirection = kEnemyMovingLeft;
   return;
}
```

I set the refcon for the object so the enemy sprite is animated as it moves. If the refcon is even, one version of the sprite is drawn. If odd, another is used. (The refcon is also used to help determine where in the enemy object list I left off when processing the aliens.)

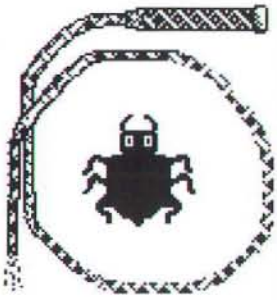


The Debugger V2 & MacNosy

by Steve Jasik

The Ultimate Macintosh Debugger

For more info
visit us at: http://www.jasik.com



Information

Control

Jasik Designs • 343 Trenton Way, Menlo Park, California 94025 • (415) 322-1386

I then increment the gNumEnemiesProcessed variable so that I'll know when I run out of "gas" for this iteration though the loop. Next, I randomly decide if this particular invader should fire. I move the unit left, right, or down, check to make sure it hasn't reached the bottom of the screen (in which case the game ends), and finally check to see if the aliens need to change direction. Note that once one enemy decides to change direction, all the others follow suit.

That's it. Not extremely sophisticated, yet tricky enough that it takes time and effort to do it right.

Draw everything

By this time, I have gathered all the input, performed all the logic, and done all of the collision detection. Now I want to draw everything so the user can see it.

All of the objects in SprocketInvaders are stored in lists. Look back at the code for GameLoop, you see that I call GameObjectListDraw for all lists maintained. Note that I pass backBuff as the second parameter to GameObjectListDraw. This background buffer is created during the initialization of DrawSprocket.

Anyone who has tried doing any computer animation is likely to be familiar with the bane known as "tearing" — when objects moving on the screen have their top halves drawn in one location and their bottom halves drawn in another. This effect takes place because the game is changing the contents of the video buffer while the computer is busy reading the video frame buffer and drawing it on the screen.

One of the best ways to avoid this ugly effect is to do all of your drawing into a back buffer, in main memory or in VRAM, then copy the entire back buffer into the frame buffer that is being drawn to the screen. An even more sophisticated method does this process while "syncing to VBL", which means that the copy is done only when the monitor's scan line has reached the bottom-right corner of the screen, and the monitor is resetting it to the top-left corner. These methods have been around for years and there are many articles and books that describe how to do them on MacOS computers. However, with the advent of DrawSprocket, all of the ugly details of doing double-buffered graphics are hidden from you, and you are left with an extremely easy API allowing you to get all the benefits of double-buffered, VBL-synced graphics without any of the annoying mess.

Listing 12: GameObject.c

GameObjectListDraw Snippet

Iterates through the given object list, drawing each object.

```
for (index = list; index; index = index->next)
{
  index->screenRect = SpriteToGWorld(index->sprite,
        index->frame, dest, index->screenX, index->screenY);
  GraphicsSetRectDirty(&index->screenRect);
  index->oldScreenRect = index->screenRect;
}
```

GameObjectListDraw calls SpriteToGWorld to copy the sprite image into the back buffer. It then calls GraphicsSetRectDirty to tell DrawSprocket that, when it is time to copy the back buffer to the front buffer, this particular area needs to be refreshed. This is an extremely important step to ensure optimal performance. When you specify the "dirty" (changed) areas of your back buffer to DrawSprocket, it copies the contents of

OpenGL® for the Macintosh™

The Industry Standard for 3D Graphics

- Port your Code from other Platforms with Ease!
- Workstation Class Performance on your Mac
- 100% OpenGL Compliant
- RAVE Hardware Acceleration Gives you the Speed you Need
- Multi-processor Capable

Checkout our Demos: www.conix3d.com
Ask us about Custom Libraries

Conix Graphics
Your OpenGL Solutions Source

Order Direct:
sales@conix3d.com
817.467.0461 voice
817.467.9452 fax
Single License $279
Commercial License $499
We offer student discounts and site licenses

Power Mac and Macintosh Developers:
FIND OUT FAST WHAT'S GOING ON IN MEMORY

THE MEMORY MINE™

- See memory allocation in any open heap at a glance.
- Easily spot memory leaks.
- Flags heap corruption when it happens.
- Works with source level debuggers to let you find memory problems fast.
- Stress applications on the fly with Purge, Compact, and Zap.
- Allocate memory at will for precise stress testing.
- Log heap data - easily document heap status over time.
- No need for source code: nothing inserted in code; no patches to the system.
- Works with 24-bit, 32-bit, and modern memory managers.

For Macs with 68020 or better. Requires System 7.0 or later.

only $99 US
Order now from Adianta, Inc.
Phone: (415)781-8052 • FAX: (415)781-8053
AOL:Adianta • Internet:adianta@aol.com
For VISA, MC, or American Express orders by mail, fax, or Internet, please include name, address, card number, expiration date, and phone number or email address.
Also available through APDA and the MacTech Mail Order Store.
for more information contact
Adianta, Inc. • 582 Market St. #911 • San Francisco, CA 94104

only those areas to the front buffer. If you move only one or two sprites, copying only the areas that have changed is far more efficient than copying the contents of the entire back buffer to the front buffer.

You might note that I am not calling DrawSprocket directly from within this routine. The reason is that I wanted to make the code as portable as possible. This is a good example of what I mean when I say you should implement your code in a flexible manner. By abstracting out the blitting of the sprite to the back buffer and the function for setting the dirty rectangles, you allow yourself the option of easily removing DrawSprocket and putting in another drawing library. This might be necessary if I wanted to port the game to Windows, where I wouldn't have DrawSprocket available.

After I've copied all of the sprites into the back buffer, it is time to finally move the image to the screen so that the user sees it. DrawSprocket has made this task extremely easy. Here is the entire code for GraphicsUpdateScreen, the last function called in GameLoop.

Listing 13: Graphics.c

GraphicsUpdateScreen Snippet

Code that gets everything displayed on the monitor.

```
DSpContext_SwapBuffers(gDisplayContext, nil, nil);
```

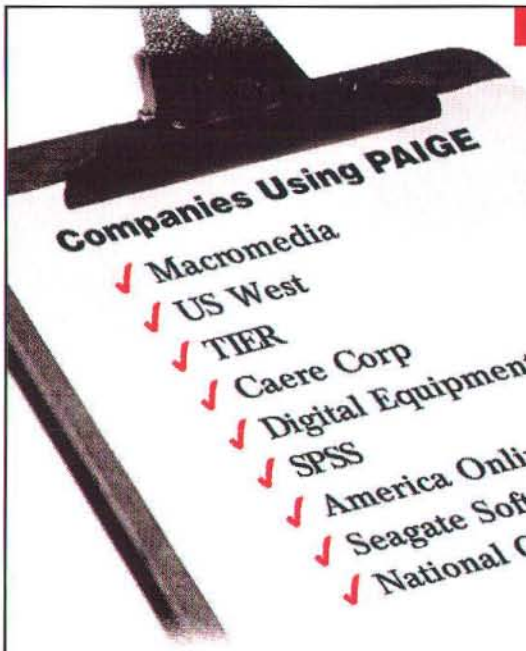
This function copies the dirty areas of the back buffers into the front (display) buffer. Not only that, but it also syncs this copy to the next VBL to eliminate tearing. DrawSprocket uses on-the-fly code generation to gain optimal performance for your copies. Without DrawSprocket, you would have to do a large amount of work to get the same performance and features you get out of this single call.

Step Six — Networking

Network support was the last feature added to the game. I didn't plan on making SprocketInvaders a networked game, but NetSprocket makes it so easy to add networking that I couldn't resist. If I had wanted bare-bones NetSprocket support, I could have had it with much less code than I ended up with in NetSprocketSupport.c. I decided it would be cool to have a chat window before the game actually begins. In retrospect, this may not have been the best design decision, since SprocketInvaders is limited to two players and there really isn't much to chat about. However, you should still be able to use the chat-related code in your game.

I already discussed initializing NetSprocket in step four. What I want to cover here are three other areas — hosting, joining, and playing. There are some other parts of handling the networking, such as what to do when the game ends, that I will gloss over. Once you understand the code for the three areas listed, you should understand the rest.

With NetSprocket, hosting a game on a network is a piece of cake. To host a game, you need to do two things. First, you need to present a dialog box that allows the user to specify the game's



For C and C++ Developers ....

If these companies are profiting from PAIGE, why aren't you?

Companies Using PAIGE

- Macromedia
- US West
- TIER
- Caere Corp
- Digital Equipment
- SPSS
- America Online
- Seagate Software
- National Geographic Society

Are you writing a . . .

- Document Viewer?
- Report Generator?
- HTML Editor?
- Electronic Book Technology?
- Hypertext Enabled Program?
- Program with Text Features?
- Multimedia Product?
- Forms Package?
- Email Application?

PAIGE will save you hundreds of precious development hours. Better yet, with PAIGE's cross-platform API you can support the most popular desktop operating systems.

DataPak Software Inc.

Internet: sales@datapak.com
www.datapak.com/~datapak
Ph: 360-891-0542
CS: 76424,3027
AOL: DATAPAK1
Fx: 360-891-0743

Call: 800-327-6703

Visit our web site to download our PAIGE demo:
www.datapak.com/~datapak

Windows 3.1 • Windows 95 • Windows NT • Macintosh • PowerMacintosh

name on the network and the network protocol(s) to use. Since NetSprocket hides the whole issue of transport protocol from you, it also presents a configuration interface to the user.

Listing 14: NetSprocketSupport.c

DoHostGame Snippet

Hosts a game on the network so that others may join.

```
status = NSpProtocolList_New(NULL, &theList);
// Do the host dialog
GetChooserName(playerName);
CopyPStr(kNetworkGameName, gameName);
password[0] = 0;
OKHit = NSpDoModalHostDialog(theList, gameName, playerName,
        password, NULL);
if (!OKHit)
  return (false);

// Now host the game
status = NSpGame_Host(&gNetGame, theList, kMaxPlayers,
    gameName, password, playerName, 0, kNSpClientServer, 0);

FAIL_OSSTATUS(status, "NSpGame_Host returned an error");

NSpProtocolList_Dispose(theList);
```

When the NSpDoModalHostDialog returns, it tells you whether or not the user hit OK. If so, you need to call the NetSprocket function that actually creates all the necessary endpoints, etc., so that others may join your game. Information gathered from the user in the dialog box is passed along to the NSpGame_Host function using an opaque protocol list. You don't need to know anything about the protocol lists or the protocol refs, except that you need to create an empty list to pass to NSpDoModalHostDialog and then to NSpGame_Host. For more information about the parameters to the two host functions, see the NetSprocket documentation available on the Web.

After I successfully host the game on the network, the DoHostGame function calls WaitForAllPlayers. I won't get into this function here, since it really doesn't make up an integral part of the game. Feel free to take a look at the code and borrow whatever you want.

Joining a game on the network is even easier than hosting the game.

Listing 15: NetSprocketSupport.c

DoJoinGame Snippet

Joins a game being hosted on the network.

```
GetChooserName(name);
password[0] = 0;

gIAmHost = false;

theAddress = NSpDoModalJoinDialog(kNBPType, kJoinDialogLabel,
        name, password, NULL);
if (theAddress == NULL)

// The user cancelled
```

```
  return (false);

status = NSpGame_Join(&gNetGame, theAddress, name, password,
            0, NULL, 0, 0);
FAIL_OSSTATUS(status, "NSpGame_Join returned an error");
```

Again, I call a NetSprocket HI function so it can enable the user to choose a network protocol and select a game to join. I assume that the host and the joiner will somehow agree on what networking protocol they are going to use.

Rather than a protocol list, the NSpDoModalJoinDialog returns an opaque address ref that is passed to NSpGame_Join. The reason is that you can join only one particular game at a time, while NetSprocket allows you to host a game on multiple networks at the same time. It is entirely possible for you to be playing a NetSprocket game that has some people using TCP/IP and others using AppleTalk. Of course, that is not possible in SprocketInvaders, since it limits the number of players to two.

The last thing I want to discuss about the networking is how to handle sending and receiving messages. Recall that I get the remote player's input from the network. Let's look at the code that actually does that.

Listing 16: NetSprocketSupport.c

HandleNetworking Snippet

Gets all network messages from the other player.

```
theMessage = NSpMessage_Get(gNetGame);
if (theMessage != NULL)
{
  DoHandleMessage(theMessage);
  NSpMessage_Release(gNetGame, theMessage);
}
```

DoHandleMessage is just a big dispatcher that handles different kinds of messages that I might get, including player input, chat message, game over, etc. Since I am interested in input messages, I will look at the code that deals with an input message.

Listing 17: NetSprocketSupport.c

DoHandlePlayerInput Snippet

Processes the player input message from the remote player.

```
if (gIAmHost)
{
  gGameKeys.redLeft = inMessage->left;
  gGameKeys.redRight = inMessage->right;
  gGameKeys.redFire = inMessage->fire;
}
else
{
  gGameKeys.greenLeft = inMessage->left;
  gGameKeys.greenRight = inMessage->right;
  gGameKeys.greenFire = inMessage->fire;
}
gReceivedInput = true;
```

Definitely not rocket science. I just fill in the remote player's keyboard state and hit a flag to let the game loop know that I've received the other player's input.

I have defined a NetSprocket message structure for player input messages. Like all NetSprocket messages, the player input message has the NSpMessageHeader structure as its first field. This header is what NetSprocket uses to deliver messages to the intended recipients, and it also contains the information necessary for the recipient to determine what kind of message it is. When I want to send a new message, such as a local input, I fill in the message structure.

Listing 18: NetSprocketSupport.c

SendInputState Snippet

Sends our input state to the remote player.

```
PlayerInputMessage  theMessage;
OSStatus status;

NSpClearMessageHeader(&theMessage.h);
theMessage.h.to = kNSpAllPlayers;
theMessage.h.what = kInputMessage;
theMessage.h.messageLen = sizeof(PlayerInputMessage);
theMessage.left = left;
theMessage.right = right;
theMessage.fire = fire;
status = NSpMessage_Send(gNetGame, &theMessage.h,
      kNSpSendFlag_Registered);
```

Here I create a message that is addressed to all players. Conveniently, this means "the other player" in a two player game, since messages sent to everyone are by default not delivered to the sender. I fill in what kind of message it is so the recipient will know how to interpret it, and how big it is so that NetSprocket will know how to deliver it. With the header information completed, all that remains is to fill in the actual message information, which I obtain from the globals (which were, in turn, filled in by the Get*Player*InputState). I then tell NetSprocket to send it, and I am done.

Without NetSprocket, adding network support for SprocketInvaders would be a daunting task, requiring a large amount of work and testing. I probably would have had to choose between using AppleTalk or TCP/IP, and probably would not have bothered with a chat screen. To be honest, it probably would not have been worth the effort.

Step Seven — Sound

I made the effort to use Sound Sprocket to localize the sounds in SprocketInvaders, just to show how easy it is to do. By localizing, I mean that I attenuate the sound's volume and left/right balance based on the position of the sound-generating object relative to the listener. You can write the localization code yourself, but why bother when SoundSprocket can do it for you?

Earlier in this article I talked about initializing SoundSprocket. All I really did was allocate sound channels, install filters in the channels, and set the listener location (effectively, the player's location) to be the bottom middle of the screen. As I play the sounds, I'll specify the location of the sound-generating objects (such as an invader firing a missile) in the same coordinate system as the listener, so that the filters alter the sounds as if the human player were inside the game.

Every time I want to play a sound, I call the SoundHandlerPlay function.

Listing 19: SoundHandler.c

SoundHandlerPlay Snippet

Plays a sound, doing 3D localization first if SoundSprocket is installed.

```
if (gSoundSprocket)
{
TQ3CameraPlacement  location;
SSpLocalizationData localization;

  // Position the sound in space. The lower left corner of the screen is (0, 0)
  location.cameraLocation.x = 640 - x;
  location.cameraLocation.y = 480 - y;
  location.cameraLocation.z = 0;

  // Orient the sound so that it is down directed down towards the listener
  location.pointOfInterest.x = 320;
  location.pointOfInterest.y = 0;
  location.pointOfInterest.z = 0;

  location.upVector.x = 0;
  location.upVector.y = 0;
  location.upVector.z = -1;

  theErr = SSpSource_SetCameraPlacement (gSource[theSound],
           &location);
  if (theErr)
    FatalError ("Failed to set the source location");
  theErr = SSpSource_CalcLocalization (gSource[theSound],
           gListener, &localization);
  if (theErr)
    FatalError ("Failed to calculate the localization");

  // We don't do doppler, since we only localize the sound at the instant it is played.
  localization.currentLocation.sourceVelocity = 0;
  localization.currentLocation.listenerVelocity = 0;

  // We set the reference distance to a reasonable number that seems to get good
  // results
  // We also provide a minimum distance or else we run into really LOUD, really
  // CLIPPED sounds
  localization.referenceDistance = 20.0;
  if (localization.currentLocation.distance < 5.0)
    localization.currentLocation.distance = 5.0;

  theErr = SndSetInfo (gChannels[theSound],
           siSSpLocalization, &localization);
  if (theErr)
    FatalError("Failed to localize the channel");
}

  // Play the selected sound immediately
theErr = SndPlay(gChannels[theSound], (SndListHandle)
         gSounds[theSound], true);
if (theErr)
  FatalError("Failed to start sound in SndPlay().");
```

If SoundSprocket is installed, use the position information of the object in the game to create a location that SoundSprocket can understand. I also need to create a point of interest, because SoundSprocket allows you to have an object project a sound in a certain direction (think of the difference in sound of a person yelling depending on whether that person is facing you or facing away from you.) SoundSprocket calculates the necessary localization information that I then pass into the sound channel, which in turn passes the information to the sound filter installed at initialization time.

That completed, I actually play the sound by calling SndPlay. If SoundSprocket is not installed, all of the steps above are skipped, and the sound is just played — always at a single loudness, always with equal stereo balance. I believe you can hear and appreciate the difference in effect if you run SprocketInvaders with SoundSprocket installed, then run it without.

Step Eight — Enjoy

Over the course of this article, I have gone through all of the major steps necessary to take an idea for a game and turn it into reality. I hit all of the major areas that once would have presented serious, possibly insurmountable, challenges for a would-be game developer. I have shown how the Apple Game Sprockets make it much easier for you to implement the most difficult aspects of your game — input, high-speed blitting, and networking — and to add cool features you might never have considered like 3D sound.

I presented SprocketInvaders in a very structured manner, but obviously things don't always happen that way in real life. Do not be discouraged if things don't happen in quite the same structured way when you sit down to write your game. Different aspects take different amounts of effort for different people. Here is an estimate of how long it took to utilize the various Game Sprockets.

- InputSprocket — under an hour.
- SoundSprocket — about an hour.
- DrawSprocket — about four hours.
- NetSprocket — two hours for the basic support, another day for the chat dialog.

The SprocketInvaders source code has some other little programming jewels that I didn't even mention. For instance, there is source to play "Red Book" audio from CDs and a set of debugging memory management functions. It can be a fantastic resource for your next game.

I know there are many cool MacOS games waiting to be written or sitting half-written in archived disks. I hope that what you've learned here will enable you to dust off those archives or sit down and start working on your game, comfortable with the knowledge that you'll be free to work on the game itself, rather than on the tedious infrastructure necessary to implement it. I hope yours is the next insanely great, totally addictive, totally cool game for the Mac.

Credit Where It's Due

This article would not exist if it weren't for the sample code SprocketInvaders, written by Chris De Salvo, with contributions from Cary Farrier, Michael Evans, Tim Carrol, and myself.

Thanks to reviewers Kathy Osborne, Chris De Salvo, Cary Farrier, and Brent Schorsch.

No llamas were injured in the creation of this article. MT

Want to suggest an article for the magazine? Send your suggestion to <mailto:editorial@mactech.com>

By Mark Chally, Covina, California

HappyFace

Using the Communications Toolbox to build an interactive, two-player game

The purpose of this article is to implement a simple, two-player, interactive game as a means to explore Apple's Macintosh Communications Toolbox. It includes methods for using the Connection Manager to send and receive information. This article builds on the fundamentals learned in the DebugTerm article, found in MacTech 12.10 (October, 1996).

Introduction to HappyFace

Where did it come from?

After DebugTerm went to press, I suggested a follow-up article to further exploit the CTB. I thought a simple game would be entertaining and instructive. Time was short, but the challenge was exciting. Of several possible game ideas, HappyFace was the clear choice, having the advantages of simplicity and entertainment value.

Long before Barney the dinosaur would be loathed by adults everywhere, there were happy faces. Surely you remember the little yellow pill-shaped guys with Kool-Aid grins, found on T-shirts, pins, and bumper stickers accompanied by "Have a Nice Day." Soon, T-shirts displayed happy faces with bullet holes in their foreheads from which blood oozed. Later, arcade games appeared in which players would use mallets to strike gophers' heads as they poked up through holes. Finally, Mike Darweesh wrote the Macintosh game "Bonk," based on the gopher games. In Bonk, players click on silly heads as they appear in a grid, before they disappear — at progressively shorter intervals. HappyFace was inspired by that weird combination of sarcasm and gaming.

How do you play it?

HappyFace is a two-player game that gives you the opportunity to smack those little yellow critters. You use the CTB to connect with your opponent. HappyFace does not care what tool is used to establish the connection, but it does expect the tool to support a means to simultaneously send and receive data using the standard CTB functions. Most, if not all connection tools do this. You earn points by striking the faces as they appear on a grid. The first player having 25 points wins.

When launched, if HappyFace cannot find the preferences file in the program's folder it poses the Connection Settings dialog (**Figure 1**).

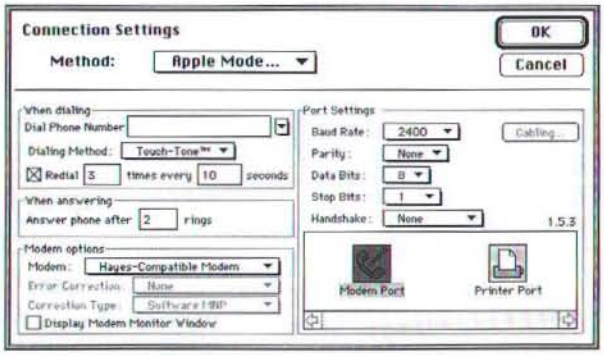


Figure 1. *Connection Settings Dialog with Apple Modem Tool's default options.*

Though chiefly compensated for Information Systems work, **Mark Chally** has derived much of his satisfaction from developing programming tools, utilities, and games — such as HexFlags DA, Flipper FKEY, ShowScreenSize FKEY, PathKey, DensePrint and Telekinetic Chess. Visit his web page at <http://home.earthlink.net/~chally> or send your ideas and comments to <chally@earthlink.net> or P.O. Box 2295, Covina, CA 91722.

If the user cancels the dialog, the program beeps and aborts. Otherwise, it creates a game window with 64 squares, eight rows and eight columns. HappyFace then waits for the user to establish a connection, which is initiated using the Connect, Listen, and Settings items on the Connection menu.

Once a connection has been established, either player may select New Game from the File menu. When the game begins, each player's copy creates five happy faces which inhabit randomly selected squares in both players' windows (**Figure 2**).

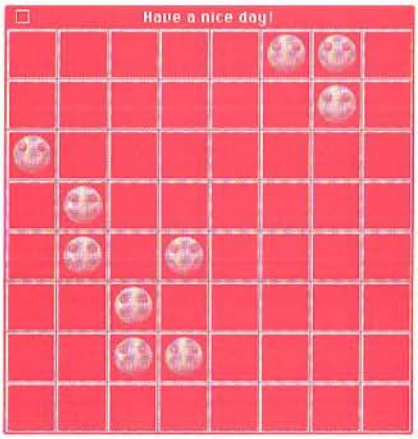


Figure 2. *The HappyFace window with no faces struck.*

Each player may then strike happy faces by clicking on them using the cursor — which becomes a mallet whenever it is positioned over the window. As the faces are struck, their features (eyes and mouths) change color and shape. Faces which have not been struck have blue features. Those struck by the player using the local copy receive green features, and those struck by the opponent receive red ones (**Figure 3**).

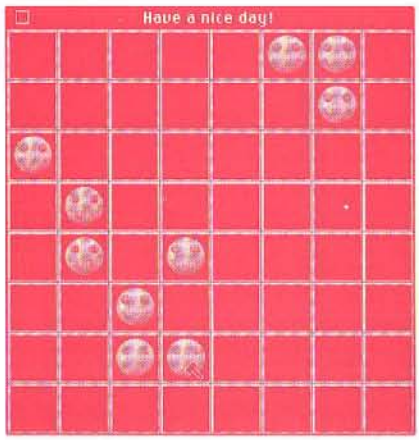


Figure 3. *The HappyFace window with faces struck by both players.*

One second after each face has been struck it dies and vanishes. The copy of HappyFace which created the face (the local copy or the opponent's) replaces it with a new one that inhabits another randomly selected square. The first player to strike 25 faces wins, at which time a dialog is displayed by each copy to inform the users of the good or bad news. When no game is in progress there is no mallet, and any faces remaining do not respond to being clicked. At any time during the connection a new game can be started, including while one is in progress. If a user quits while connected, the connection is closed. If a connection is closed, any game in progress ends.

CTB Recap and Comparison to DebugTerm

What is DebugTerm, and why read about it?

In the DebugTerm article, I implemented a C source file that could be compiled into a program under development to export debugging information through a CTB connection. It made a good introductory project because it required minimal CTB services. If you have not read it, I strongly suggest that you do since it provides an introduction to the CTB and some base code for HappyFace. In short, this article builds on that one.

Where did DebugTerm fall short?

DebugTerm is a good start, but not a complete program. Serving as a debugging tool, it takes shortcuts in opening and closing a connection that a user-friendly program should not. Secondly, it only sends information, it does not know how to receive it. Additionally, it only initiates a connection using CMOpen() to connect with another device or program; it does not know how to use CMListen(). Finally, it asks the connection tool not to use windows or menus, thus requiring no event processing. HappyFace addresses these new issues.

Making a Complete Application

Isolating connection logic

HappyFace behaves as a CTB program should. It initializes the CTB and creates and configures a connection record at launch time. However, CMOpen() is not executed until the user wishes to open a connection. In contrast, DebugTerm executes CMOpen() immediately after creating and configuring the connection record. DebugTerm also disposes of the connection record at the same time that it closes the connection. HappyFace waits until quitting time to dispose of the connection record, at which time it also saves the configuration. For this reason, the process of opening and closing connections is simple and reliable.

It pays to CMListen

Besides isolating the initialization from opening and the disposal from closing, HappyFace also adds a new and important way to connect: CMListen(). Some tools require one party to use CMListen() while waiting for the other to initiate the connection using CMOpen(). For example, in response to CMListen(), the Apple Modem Tool waits for a telephone call from another modem. The AppleTalk ADSP Tool registers itself on the network and waits for a connection request from an ADSP Tool using CMOpen().

Both expect the application to respond to CMOpen() by accepting the connection using CMAccept(). The Serial Tool does not support CMListen(). It uses CMOpen() to establish a connection through the serial port you choose. Unlike most tools, it does not care about whether anything is at the other end of the serial cable. HappyFace does another important thing that DebugTerm did not do — it polls the connection for incoming messages using CMRead().

Handling the interface

In addition to those details, HappyFace also accepts a new realm of responsibilities. It maintains a Connection menu that allows the user to decide when and how to choose and configure a tool, and establish a connection. This means that HappyFace has the duty of maintaining these menus, in addition to the events, most of which it passes along to the connection tool via CMEvent(), CMActivate() and CMResume(). These events may also be generated for additional windows or menus that HappyFace permits the tool to create and maintain. Certain events can be identified as belonging to the tool and passed along. HappyFace detects and delegates these.

Encapsulating CTB code

Virtually all of these activities, which relate solely to maintaining a CTB connection, are contained within CTB.c. HappyFace, merely *an example* of a CTB application, can take advantage of this by installing a few MENUs, passing along important events, and making a few simple function calls. The ease with which CTB.c can be used is apparent since CTB.h profiles only a few simple function calls.

Encoding a game message

There are three game actions requiring messages: start a new game, create a face, and strike a face. Communication logic is simplified by stuffing an entire message into a single byte of the form AAHHHVVV.

The first two bits represent the game action, the next three represent the horizontal position if applicable, and the last three represent the vertical position if applicable. Some bitwise manipulation is required when sending and receiving the message, but the utility of treating a single incoming character as an entire message greatly simplifies things.

Test Spin

Before exploring the workings of HappyFace, let's watch the game in action. There are many ways to connect using the CTB, but you should consider the following points that you may find remarkably similar to those of the DebugTerm article.

Means of Connection

If you have an AppleTalk network or intend to demo HappyFace on a single Mac, I recommend that you connect using the AppleTalk ADSP Tool. Otherwise, I recommend the Serial Tool. If you choose the Apple Modem Tool you can connect by modem, but probably will not have the opportunity to see both copies of HappyFace side-by-side. You may choose to use another tool, in which case I am interested in hearing about your experiences.

Using the Serial Tool you can connect two Macs, or even connect a Mac's modem port to the same Mac's printer port. To use the Serial Tool, you will require a *null modem cable*. This is the cable used for most printers and to communicate with a Newton by serial port. It is up to you to get the connection right, but if it works between terminal programs, it should work for HappyFace.

If you choose the AppleTalk ADSP Tool, you can connect between two Macs on a network, or from one Mac to itself. To connect a Mac to itself with the AppleTalk ADSP Tool *no physical connection is necessary,* but be sure AppleTalk is turned on in the Chooser or the tool will not load. If you need a copy of the AppleTalk ADSP Tool, you can find it on one of Apple's FTP sites. You may also find what you need on ETO or the CTB SDK. HappyFace needs the Communications Toolbox, 32-bit QuickDraw with at least a 16-color monitor setting, and the Sound Manager. It would be a good idea to use System 7.5 or later since HappyFace does not use Gestalt to check these.

Connecting to Yourself

If you choose to connect the Macintosh to itself, either by using the AppleTalk ADSP Tool for a virtual hookup, or Serial Tool from modem to printer port, you should keep one thing in mind. *To connect a Macintosh to itself, you must use asynchronous calls.* See the diatribes in the DebugTerm article about asynchronous calls and completion routines. HappyFace also has an *"empty"* completion routine. To allow a connection between two copies on the same machine, I compiled HappyFace using asynchronous calls. Though many readers without two Macs may wish to take advantage of this feature, you will find HappyFace awkward to play as a two-player game when both copies are on the same machine. Also, the HappyFace I compiled is a fat binary so that it will run native on PowerPC machines.

Bang Away!

You will need to use two instances of HappyFace. Ideally, these instances would be run on separate machines. However, if you want to run both copies on the same machine, simply make an additional copy *in another folder* and launch each separately. Preferences are stored in the application's folder. If you have already run HappyFace, you may wish to throw away the file named "HappyFace Prefs."

Launch one copy of HappyFace. If it finds no preferences file, it poses the Connection Settings Dialog before allowing you to do anything else. If not, select Settings... from the Connection menu. In the dialog, use the Method popup to choose the connection tool you wish, such as the Serial Tool or the AppleTalk ADSP Tool (**Figure 4**).

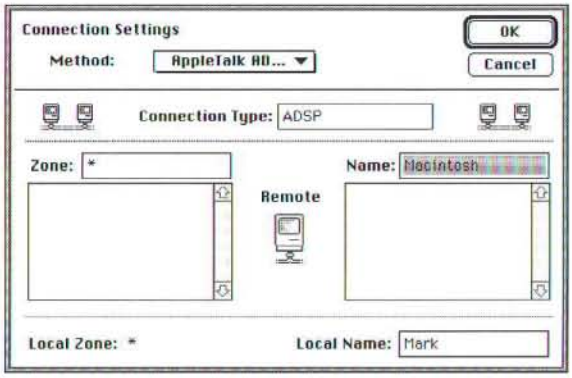


Figure 4. *Connection Settings Dialog with the AppleTalk ADSP Tool's options.*

If you chose the AppleTalk ADSP Tool, you may choose a name for the machine by typing what you wish into the Local Name: field in the lower-right-hand corner. It is probably best not to change the Connection Type field, but if you do, both instances of HappyFace must have the same type. Press the OK button.

If you chose the Serial Tool, you can select the port settings that you require. The ones displayed are probably the best (**Figure 5**).

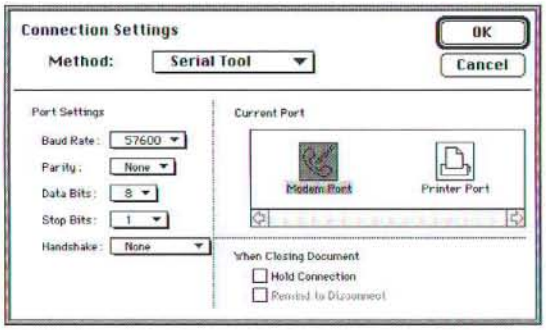


Figure 5. *Connection Settings Dialog with the Serial Tool's options.*

Regardless of how you set them, they should all be set the same for both copies of HappyFace, and the Current Port should be set to whichever port your cable is plugged into. Press the OK button.

If you chose the AppleTalk ADSP Tool, choose the Listen item from the Connection menu. You can tell that HappyFace is listening for a connection because the Listen item now reads Stop Listening, and the other items on the Connection menu are dimmed.

If you chose the Serial Tool, choose the Connect item from the Connection menu. You can tell that you are connected (to the serial cable) because the Disconnect item is now enabled and the other items on the Connection menu are dimmed.

Now you are ready to connect the second copy of HappyFace to the first. Launch the second copy. If you chose the AppleTalk ADSP Tool, choose it again from the second copy. You should see the name of the first, listening copy in the scrollable list on the right-hand side under the Name field. You may need to find the AppleTalk Zone that the first copy is in from the list on the left-hand side, under the Zone field. Select the name of the first copy, and press the OK button.

If you chose the Serial Tool, select the same settings as in the first copy, except the modem or printer port as necessary. Press the OK button.

Now select Connect from the Connection menu. The two copies of HappyFace should be ready to start a game. You can tell they are connected because the New Game item on the File menu, and the Disconnect item on the Connection menu are now enabled, and all the other items on the Connection menu are dimmed. If you select New Game from the File menu ten happy faces should appear in each window.

Use the cursor (mallet) to strike faces in each copy. You should see something like **Figure 3** when faces are being struck from both copies. Notice how the features of a face struck locally become green, but those of one struck by an opponent turn red. Notice that HappyFace allows one second to pass before the face you struck "dies."

Normally the game ends when a player strikes 25 faces. Quitting either copy by selecting Quit from the File menu or closing the HappyFace window breaks the connection. Selecting Disconnect from the Connection menu also ends any game in progress. Continue to play as many games as it takes to get a feel for it.

Source Code Walk-through

HappyFace was written in C, using Metrowerks CodeWarrior Integrated Development Environment (IDE) 1.7. I used two project files — one for 68k, and one for PowerPC. If you use Symantec, MPW, or some other development system, you should not have much trouble creating project files. You will either want to print a copy of the source code or view it on your monitor alongside the article.

Overview

This project is divided into three separate code domains. The file CTB.c isolates the CTB code so that it can be used, much as a library would, along with CTB.h. It should require little modification to be included within another project. It contains private data structures and functions specific to the operation of the CTB. CTB.h contains useful constants and function prototypes to #include in other source files. Game.c contains functions and private data structures that are specific to the game logic, but have no specific knowledge of the CTB. Game.h contains prototypes for the Game.c functions that Main.c needs to call. Main.c contains the "boilerplate" code necessary to launch the program, handle simple events, maintain the cursor and menus, and quit.

Header files

CTB.h is similar to DebugTerm.h. In addition to function prototypes, it contains shared resource ID's, and literals for parameters and common error codes. I shamelessly kept the

error codes from DebugTerm, although I could have developed a more extensive set. The literals, such as kAsync, help make the code readable. Game.h has prototypes to game functions that are called by Main.c.

CTB.c

This file contains C code necessary for Communications Toolbox connections. You are already familiar with most of it, having read about DebugTerm which was extended from DebugTerm.c. We now concentrate on new or modified functions.

It begins with a #include to CTB.h, then defines its literals and globals. One of the "new" features is a group of Universal Procedure Pointers (UPP), such as gCMCompletion, for PowerPC compatibility.

CMWriteProc(), formerly CMSendProc(), is unmodified. However, you should inspect it closely since I originally uploaded an outdated version of CMSendProc(), which I have since replaced with a more correct version.

ConnEvent() handles events for a connection tool window. It is called from the event loop to give the connection tool the first opportunity to handle the event. CTB tools poke their own handles (e.g. ConnHandle) into the refcon fields of their windows. Therefore, ConnEvent() finds the window associated to theEvent. Next, it makes sure it has a valid WindowPtr and ConnHandle. If not, it returns false, else it checks to see if the window's refcon is the same as the ConnHandle. If they are not the same, it returns false. Otherwise it returns true, indicating that the event was handled, after calling CMEvent() to delegate the event to the connection tool, which handles the event if appropriate.

ConnActivate() informs a connection tool of an activate event. Called from the place in the event loop where activate events are trapped, it passes the message on to the connection tool, indicating whether the window is being activated or deactivated. ConnActivate() saves the current GrafPort and sets it to the given window before calling CMActivate(), then restores the current GrafPort. I found this behavior, taken from sample code in *Inside CTB*, odd. I would have expected the WindowPtr to be passed as a parameter instead.

ConnResume() is structurally identical to ConnActivate(). It informs the connection tool of a resume (or suspend) event. The resuming parameter indicates whether the application is being suspended or resumed.

putc(), provided for simplicity, calls CMWriteProc() to send the character it receives as a parameter. It returns whatever value it receives from CMWriteProc().

CMReadProc() attempts to read *theSize characters by calling CMRead(). If gConn is not nil, it returns the value obtained by calling CMRead(). If there is no connection record, it passes back 0 as *theSize. This version of HappyFace always issues CMRead() calls synchronously to make the article simpler. Next time, we will need to expand its capability. CMReadProc() is a Pascal function so it can be called by the Terminal Manager or the File Transfer Manager in a broader implementation.

getc() tries to read a single character by calling CMReadProc(). If it finds none, it places a NULL in c. It returns whatever error CMReadProc() encounters.

InitCTBStuff(), essentially the same function as its DebugTerm counterpart, additionally generates UPPs for the CMCompletion(), CMWriteProc(), and CMReadProc() by calling the appropriate managers' functions.

ChooseConnection() has been modified to behave better in a modeless program environment. Now kNothingChosenErr is only returned if cancel is pressed (so that it can be trapped at startup when no preferences are found). It still returns noErr if the current tool is reconfigured or a different tool is selected, since we do not need to care because we never allow this dialog to be posed while connected. In all other cases, it returns whatever CMChoose() does.

NewConn() is one of three functions that replaces the functionality of DebugTerm's OpenDebugTerm(). NewConn() handles most of its functionality, making it almost identical. NewConn() does not call WritePrefs() or CMOpen() where OpenDebugTerm() did. These tasks are left to DisposeConn() and OpenConn(), respectively. It does a few additional things, however. It initializes gFlags to false, which, when passed to CTB functions, (currently, according to *Inside CTB*) indicates that we do not wish to send end-of-block indicators. It also passes the async parameter through to the global variable gAsyncCalls and sets gPrevStatus to an impossible token value, which will cause the status to be re-evaluated in IdleConn(). Finally, it calls GetMenu() for the handle to the Connection menu, which was installed in the main program. If NewConn() encounters an error, it returns it, else it returns noErr.

OpenConn(), called when the Connect item is selected from the Connection menu, calls CMOpen() to open a connection. First, however, it verifies that the connection record exists. It returns whatever value CMOpen() returns.

ListenConn() is called when the Listen item (or Stop Listening) is selected from the Connection menu. It first verifies that a connection record actually exists, and if so, it calls CMStatus(). If CMStatus() indicates that the tool is already listening for a connection, it calls CMAbort() to stop listening, else it calls CMListen(), instructing the tool to wait indefinitely (-1) for a connection. Unlike most functions which can be called synchronously or asynchronously, HappyFace calls CMListen() asynchronously regardless of the value of gAsyncCalls. It does this because some tools, such as the AppleTalk ADSP Tool, provide no opportunity to stop listening when asked to wait indefinitely. Using the asynchronous call allows the user to select Stop Listening, which causes ListenConn() to call CMAbort(). Some tools, such as the Serial Tool, do not support CMListen() at all, and return cmNotSupported. If an error occurs, the error is returned, else noErr is returned.

CloseConn() is one of two functions that split the duties previously performed by CloseDebugTerm(). When the user selects Disconnect from the Connection menu, it first verifies that the connection record exists, then verifies that the connection is open or opening. If so, it calls CMClose() to drop the connection.

Want to share a tip with the community and get paid for it? Send it in to <mailto:tips@mactech.com>

MAC PROGRAMMERS

Considering a move to sunny South Florida?

SIRS Inc., an established educational publishing company and PC software developer located in Boca Raton, Florida, has an opening for a MAC PROGRAMMER in our Computer Department.

This position requires a minimum of 2 years recent MAC programming experience with C or C++. Ideal candidate must have a BS in Computer Science.

Excellent Benefits Package
Non-Smoking Facility
Pre-Employment Drug Screen Required

Qualified candidates may mail or fax resume with salary history to:

SIRS Inc. • The Knowledge Source
P.O. Box 2348 • Boca Raton, FL 33427-2348
Fax# (561) 994-2014

The Knowledge Source SIRS Since 1973

EQUAL OPPORTUNITY EMPLOYER

PARALLEL SOFTWARE

CHICAGO AREA

Parallel Software is looking for software engineers who are serious about component based **OPENDOC** development. We are interested in Macintosh or Windows people with the following experience:

"C" or "C++", Pascal, OPENDOC
Newton, NDK, or JAVA
MacApp,Code Warrior, MacToolbox
Oracle, Sybase, or SQL Databases

Please write or phone:

Mr. John D. McMahon
Parallel Software, Inc.
608 S. Washington, Ste. 101
Naperville, Illinois, 60540
Tel (708)369-0100
Fax: (708)355-7870
psgmcmahon@aol.com.

MacTech Magazine is your recruitment vehicle

When you need to fill important positions at your company, MacTech Magazine is the consistent choice of companies across the country for hiring the best qualified Macintosh programmers and developers. Let MacTech Magazine deliver your recruitment message to an audience of over 27,000 qualified computer professionals.

Call Ruth Subrin at
805/494-9797

Visit MacTech Magazine's Web site!
http://www.mactech.com

MacRegistry™

Developer Job Opportunities

If you are a Macintosh developer, you should register with us! We have a database that enables us to let you know about job opportunities. When we are asked to do a search by a client company the database is the first place we go. There is no charge for registering. The database service is free. Geographic Coverage is nationwide.

Marketability Assessment - To get a specific feel for your marketability send a resumé via Email or call. You may also request a Resume Workbook & Career Planner.

Discreet - We are very careful to protect the confidentiality of a currently employed developer.

Scientific Placement is managed by graduate engineers, we enjoy a reputation for competent & professional job placement services and we are Mac fanatics.

1-800-231-5920 • das@spi.com • Fax 1-800-757-9003
http://www.scientific.com

Scientific Placement, Inc.

MT, Box 19949, Houston, TX 77224, 713-496-6100 Fax: 713-496-0373
MT, Box 71, San Ramon, CA 94583 510-733-6168 beth@spica.bdt.com
MT Box 202676, Austin, TX 78720-2676 512-331-0302 lej@zilker.net
AppleLink: D1580; Compuserve: 71250,3001; AOL: davesmall

DisposeConn() is the second of the two functions that split CloseDebugTerm()'s duties. Called before HappyFace exits, it first verifies that the connection record exists, then calls WritePrefs(). Once the preferences have been written, it closes the connection by calling CloseConn(), unlocks the connection handle, and calls CMDispose() to deallocate the connection record and any other data the tool may be using. Finally, it sets the connection handle to nil, and reports noErr. DisposeConn() does not report any error unless there is no connection record, because it is only called at application exit. In fact, it calls CloseConn() without checking to see if connected. It could check that first, and it could also report any errors returned by CloseConn() or CMDispose().

IdleConn(), called during each iteration of the event loop, performs periodic tasks necessary to support a connection tool. It begins by verifying that a connection record exists, and if not, it disables the connection menu items. Otherwise, it begins by calling CMIdle(), which gives the connection tool its housekeeping opportunity. Next, it checks the connection status. If an error occurs, it returns it, else if the tool has sensed an incoming call (cmStatusIncomingCallPresent) it calls CMAccept() to establish a connection. If the connection status has changed, it updates the Connection menu accordingly, updates the online parameter, updates gPrevStatus and returns any error that occurred.

DoConnMenu(), called by the main application's DoMenu() function when a connection menu item is selected, calls the appropriate CTB.c function. It doesn't bother to report or return errors, though you may wish to. It beeps in the case of an error, except kNothingChosenErr, returned by ChooseConn(). This is because kNothingChosenErr results from pressing the Cancel button, which is fine when resulting from a menu selection.

Game.c

This file contains functions to control the game activity, but no code specific to the CTB. It begins with a few groups of literals. Those in the first group denote spacing and graphics proportions. Those of the second indicate whether a square is occupied, and if so, what condition the occupying face is in. Those in the next control the bounds of the game logic. Following that are the codes for the possible game actions. Messages sent through the connection have these codes in the first two bits to indicate that a face has been made or struck, or that a new game has been started. Being a two-bit code, the possibilities are 0-3; 0 is skipped to avoid confusion with the NULL character. The literals in the last group are resource IDs.

Next in Game.c is the struct used to store information for a square.

```
typedef struct {
  short occupant;     // kEmpty, kHealthy, kStruckLocally, kStruckByOpponent
  long age;           // TickCount() at status change
  Rect boundsRect;    // Bounding rectangle for gSquare
  Boolean madeLocally;// Keep track of whether made locally
} contents;
```

The occupant field contains a square's status. The age indicates at what TickCount() the square's status last changed. We want a face to take one second to die, so we have to watch the time. boundsRect holds the local coordinates of the square's rectangle. madeLocally indicates whether the local copy or opponent's copy created the face.

In the interest of fairness, we make sure that each copy maintains a fixed number of faces. When a face is created, it takes a variable amount of time to transmit the creation message to the opponent's copy. Using the convention introduced here, the players' individual copies effectively take turns. Thus, each player has a "net equal time delay" in the sum of the differences between the times at which faces are created and become visible.

Next is a group of globals, kept private to game.c using the static keyword. These are some of the most notable globals: gPixMapHand is an array of the pixmaps used to store the different drawn faces so that they can be copied to the game window. gMadeCount is used to represent the number of currently-living faces a copy has created. gLocalStruckCount and gOpponentStruckCount are used for keeping score. gSquare is the array of square contents. gStrikeHand and gSndChannelPtr are used to play the striking sound. gPicRect is a Rect with its top-left corner at the origin and the height and width of a square.

PlayStrikeSound(), called when a face is struck, verifies that the handle to the sound and pointer to the sound channel are valid, then calls SndPlay() asynchronously.

ShowScores() is called when a game ends to display the scores to each user. It counts the points up first to see if the game was actually played. If faces have been struck, it converts the numbers to strings using NumToString(), uses ParamText() to preload text for the Dialog Manager, then calls Alert().

DrawSquare() draws the appropriate image in the appropriate place on the game window. First it assures that the parameters, h and v, are within the valid range of rows and columns, then it sets the current GrafPort to the game window after saving the old GrafPort. It calls CopyBits() to copy the contents from the appropriate pixel map we already created to the appropriate place in the window. Finally, it restores the current GrafPort.

NewGame(), indirectly called when either player selects New from the File menu, initializes the game status variables, sets each square's contents as empty, and draws each square.

MakeGWorld(), borrowed from the *Dave Mark* article, calls NewGWorld() to allocate a fresh default GWorld to store squares' graphics in.

NewGameWindow(), called at application launch, initializes everything on which the game's logic rests. It begins by seeding the random number generator with the long integer which represents the current date and time. Then it attempts to create a new sound channel and load the SND resource for the strike sound, which it moves into high memory and locks if successful. Note that the sound playing code does not attempt to play the sound if either of these could not be allocated. Next, it calculates the size and position of the window, calls NewCWindow(), fills the gSquare array with empty squares, and calculates their rectangles.

For each of the four graphics used to draw squares, it loads the pixel maps and draws them into their own separate GWorlds. Most of this was also taken from *Dave Mark*. Finally, it calls ShowWindow() to make the new window appear on the screen.

KillStruckFaces() is called by the IdleGame() function each time through the event loop. It looks at each square and determines if the square contains a face that has been struck by a player. If so, and if it was struck more than one second ago (kDyingTime), it makes the square empty, assigns the time of vacancy, redraws the square, and if the face was made by this copy it decrements the number of living faces made by this copy. Because gMadeCount will be less than kMaxMade, this will result in the eventual creation of a new face.

UpdateGameWindow(), called in response to an update event, saves the current GrafPort and sets it to the game window. Next it finds the Rect bounding the update region and uses it to decide which squares are in the region. After it has determined what range of rows and columns it needs to draw, it calls BeginUpdate() to keep from generating additional events, and draws them top-left to bottom-right. It calls EndUpdate() to cancel BeginUpdate() and sets the current GrafPort back to the saved one.

ShowStruckFace() is called when a face has been struck. It receives the horizontal and vertical location of the face, and whether the strike was local or by the opponent. It uses those parameters to update the square's information and redraw the square. Before returning, it plays the strike sound.

SendGameAction() assembles a game message and sends it. The message has three parts. The rest of the communication logic is greatly simplified by stuffing an entire message into a single byte. The first part is a two-bit action indicator, the next three bits indicate the horizontal component of the square's location, and the final three indicate the vertical component of the square's location.

First it loads the action indicator into the character, then shifts three bits to the left so that it can add in the horizontal component. Then it shifts three more bits to the left and adds the vertical component. Finally, it calls putc(), whose return value it also returns, to send the character.

SendNewGame() asks SendGameAction() to send a message with the kActionNewGame indicator through the connection (the horizontal and vertical components do not matter).

HandleOpponentStrike() is called in response to having received a message which indicates that the opponent has struck a face. First, it increments the opponent's score. Next, if the opponent has reached the winning score, it stops the game. Then, if the face is actually healthy, it calls ShowStruckFace() to change its status and redraw it. It always gives the opponent credit, but does not always change the square's status. It is theoretically possible for both players to strike a face "at the same time" and for the messages to "cross in the mail." Instead of using a tie-breaker to resolve the conflict, it gives both players credit. It does not draw in response to this case, because that would cause a face to change from green to red or red to green.

HandleWindowClick() is called when a mouse click in the game window is detected. First it determines which square, if any, was clicked, by reducing the Point to local coordinates and dividing by the pixel offset between squares. As an added protection, it verifies that the calculated value is in range, to avoid addressing a non-existent array member. If the row and column are valid and there is a healthy face in the indicated square, it uses SendGameAction() to inform the opponent of the successful strike. If no error is returned, it increments the player's score and compares it to kWinningScore. If the score has exceeded the winning score, the game will be stopped. It calls ShowStruckFace() to indicate to the user that the face has been struck.

ShowFace() puts a healthy face on a given square. It sets the square's status to indicate a healthy face, sets the age to TickCount(), and sets madeLocally according to the parameter that it receives. Finally, it calls DrawSquare() to draw the face in the game window.

MakeFace() is called when IdleGame() determines that it must place another face on the window to meet its quota, kMaxMade. First it randomly chooses a row and column, repeatedly, until an empty square is found. Next it informs the opponent's copy that it has created the face, and calls ShowFace() to show the user. Finally, it increments gMadeCount, the number of currently-living faces that have been made by the local copy.

HandleGameAction() dispatches game actions which have been received to the appropriate function, according to the action type. In some cases it starts or stops a game. Therefore, it passes back the resulting value of inProgress.

GetGameAction() is called by IdleGame() each time through the event loop. It determines whether a game message is waiting, and if so, dispatches it. First it calls getc() to read the character. If the character is not NULL and there is no error reading, it processes it. It extracts the action indicator by rotating a copy of the byte 6 bits to the right. Next it rotates another copy three bits to the right and looks at the least-significant three bits, which become the horizontal component. The least-significant three bits of the original become the vertical component. Finally, it delegates the interpretation of the message to HandleGameAction().

IdleGame() gets called once from each iteration of the event loop. It first calls GetGameAction() to dispatch any opponent actions waiting. Next it calls KillStruckFaces() to remove any faces struck more than one second before. If the game is still in progress and the number of faces is less than the quota, it calls MakeFace().

Main.c

Activities not specific to communications or game logic are conducted within this file. That leaves calling the procedures necessary for initializing the program, dispatching events, and quitting. Since most of us have seen this repeatedly, we will brush over the bulk of the code,

discussing only the unusual or confusing parts of this file.

It begins by defining literals and usual globals. Some globals of note are gHammerHand, which points to the mallet cursor; gGameInProgress, which indicates whether a game is being played; and gOnline, which indicates that a connection is open.

ExitNow() is used as an emergency exit, while DoQuit() is called when a user chooses Quit from the File menu. Both call DisposeConn(), which makes sure the connection is closed if possible. DoQuit() also shows the scores if a game was in progress, because there will be no chance for the event loop to detect that the game ended.

With the exception of MBAR, MENU, PICT, SND and ALRT IDs the application source code is oblivious to the resource file's contents — not that I recommend ripping out DITL resources or doing anything else that would confuse the resource manager or the Finder.

InitApp(), in addition to calling SetupMenus(), also does a few other things. It allocates the hammer cursor's structure, moves it in to high memory, and locks it. It calls NewConn() to install a connection record, and if it fails, beeps and quits. It calls NewGameWindow() to create the game window and initialize supporting game logic, then finally initializes Main.c's own globals, gOnline and gGameInProgress.

DoMenu() handles menu selections in the usual way, with some exceptions. Since HappyFace adjusts the cursor, it provides an arrow cursor before putting up the About box. It sets gGameInProgress to true when a new game is started. Notice that it calls DoConnMenu() to delegate a selection from the Connection menu. It does this because it knows the menu's ID. There may be a less application-specific way to sense and handle Connection menu selections. Perhaps ConnEvent() could sniff them out instead.

AdjustCursor() was one of the more pesky details of this project. Called from various places in the event loop, it changes the cursor, if appropriate. According to some Apple documentation, you can set it up so that it only gets called when the mouse travels outside a given region, which is how HappyFace was written. You may notice that it gets called in a couple other places to work around various interface problems apparently not anticipated when MultiFinder was developed. I recommend maintaining the cursor at every opportunity from the event loop, which is what later Apple documentation seems to suggest. This seems to spread less code around without effecting performance.

Main() calls the initialization functions and runs the event loop. It uses gameWasInProgress to detect when gGameInProgress changes, and mouseRgn to pass to WaitNextEvent(). It calls AdjustCursor() if a game starts or stops, because we do not want to show the mallet if a game is not in progress. When a game ends, it calls ShowScores().

Next, it calls IdleConn() to give time to the connection tool. If it finds that the connection has been severed, it stops any game in progress, because a game cannot be played with no connection. It calls IdleGame(), regardless of whether a game is currently in progress, to allow Game.c to handle its own details, including removing dead faces.

For each event returned by WaitNextEvent(), it first gives ConnEvent() an opportunity to handle it by passing it to the connection tool if appropriate. If ConnEvent() cannot handle the event, IdleConn() continues its attempt to dispatch the event. Any activate event is delegated to ConnActivate(). If an update event is for a window other than the game window, it is ignored, otherwise it is passed to UpdateGameWindow(). A keypress means nothing to HappyFace unless it is a command-key combination, which is passed to DoMenu().

The mouse events are handled in the usual way. HandleWindowClick() is called for any click in the game window's content region, if a game is in progress. This responds to the user striking a face. Suspend and resume events are trapped and passed to the connection tool through ConnResume().

HappyFace.rsrc

HappyFace.rsrc features the usual cast of characters. Star roles are filled, such as the ALRT and DITL resources used by ShowScores(), the PICT resources used to draw happy faces, and the *menus used both by* CTB.c and Main.c. Supporting characters are such as the SND resource that plays when a face is struck, SICN *resources that spice up CTB menus*, and the CURS resource used to show the hammer when faces can be struck. Atmosphere is provided by still other resources that provide icons and bundle-oriented information, version information, and the about box props.

With the exception of MBAR, MENU, PICT, SND and ALRT IDs the application source code is oblivious to the resource file's contents — not that I recommend ripping out DITL resources or doing anything else that would confuse the resource manager or the Finder.

Conclusion

What have we learned?

In this article, we explored a complete application using Apple's Communications Toolbox. We created a simple, yet entertaining game to exploit key features of the Connection Manager. Building on a previous article, we have learned to listen for a connection, poll for data, and handle interface-related concerns such as building and maintaining a connection menu and delegating the appropriate events to the

connection tool. We found that most of the tasks of maintaining a connection can be hidden from the main program. We scratched the surface of creating an interactive game, though elements of a serious, error checking application have been omitted.

Where do we go from here?

The next step is to develop a formal protocol. Such a protocol would detect transmission errors such as messages that do not reach their destinations, arrive incomplete, or arrive garbled. Such an undertaking will be the subject of the next article.

If you have comments or questions regarding this article, feel free to send them!

References

The following is a list of publications used to produce HappyFace.

- Chally — Chally, Mark. "DebugTerm: A Poor Man's Debugging Terminal." *MacTech Magazine* 12:10, October, 1996.
- Inside CTB — Apple Computer, Inc. *Inside Macintosh® Communications Toolbox*. Addison Wesley, 1991.
- Inside Mac Files — Apple Computer, Inc. *Inside Macintosh® Files*. Addison Wesley, 1992.
- Mark — Mark, Dave. "Color Animation, Part II." *MacTech Magazine* 11:05, May, 1994.
- Surfer — Alex Kazim. "Surfer" (Sample application program with Pascal source code) Essentials•Tools•Objects Vol 13. Apple Computer, 1989.

Want to subscribe to MacTech Magazine? Contact <mailto:orders@devdepot.com> call 800-MACDEV-1 or 805-494-9797, or fax 805-494-9798.

In the dark?

Shed some light on your network programming with EtherPeek™.

Why develop your networking applications in the dark?

Use **EtherPeek Lite**, Ethernet protocol analysis software, to test and debug your software and hardware communications in minutes instead of hours!

- Check for protocol compliance.
- Use hundreds of built-in decodes. (IP, IPv6, IGRP, Netware IPX/SPX, NetBEUI, NetBIOS, DECnet, AppleTalk)
- Develop custom packet decoders.
- Filter packets during or after capture.
- Test device reactions to specific packets.
- Customize or alter packets for transmission.
- Generate traffic to test varying loads.
- Write your own plug-in modules.



ORDER TODAY!

Mention This Ad For Your Special US $399.00 EtherPeek Lite Price.

CALL: 800/466-AGGP or (1)510/937-7900
FAX: (1)510/937-2479
EMAIL: eplite-info@aggroup.com
WWW: http://www.aggroup.com/eplite/



By Chris Stasny

The Language of Resources

A common sense introduction to the Resource Manager

Say Again?

I discovered a new version of English when I worked the oil fields of Trinidad years ago. Part of this dialect's uniqueness was due to the influence of multiple cultures on the island — notable for its heavy and complex accent. Eventually, I learned how to respond to things like, "And me anudder spanner. Dees one's too slight." (He wanted a larger wrench.)

Some years later I discovered that the Macintosh contained its own rendition of English, where common terms were linked in new ways to form powerful commands. This is clearly demonstrated in Resource Manager calls. Terms like USERESFILE and SETRESATTRS turn out to be everything expected and more. This article will focus on some common resource handling operations. While our syntax will be that of FutureBASIC II, the toolbox calls are so easily recognizable that programmers of all linguistic persuasions should be able to follow along without difficulty.

Resource Files

We all know that our application's resource fork is open and available. Some of us understand that the System's resource file is also open and available. In real life, there are usually between twenty and a hundred resource files open at any given moment, on any given Mac. Remember all of those font files and sound files that you dumped into the System Folder?

The layers of resources are guided strictly by the order they were opened via commands like OPENRESFILE or OPENRFPERM. If you think of your Mac as an oil drum, the System file is thrown into the barrel first and always resides at the bottom. It is followed by a plethora of extensions and additions that come online during startup. We cannot change this order. When we open a resource file, it is always thrown on top. [There are some exceptions. Most notably, CTB tools are loaded below the System resources. -ed.]

Application Heaps

When an application is double-clicked into life, a divider is used to segment the barrel in such a way that the application's resources are isolated from those of other programs. This divider does not extend all the way to the bottom of the drum, where system resources remain pooled and available for all to use. This means that resource files opened by that application can look down upon both application and system resources without ever being aware of items (owned by other applications) on the other side of the divider. This segment of the barrel is known as the application heap.

The search pattern here is important. If we ask the Resource Manager for a chunk of bits, it will respond by searching the top levels of information in our application's quadrant of the barrel. Each failure forces it to submerge deeper into the drum until it eventually finds itself scanning the System resources at the bottom. It is not possible to change this order. At best, we can tell the Resource Manager to start at a deeper level and ignore some of the sludge that has floated to the top. We do this with CALL USERESFILE(resRef).

After USERESFILE, we have effectively chopped off the top portion of the barrel, and items opened after the target file become invisible to the system. This procedure, when improperly used, often sends a program into a death spiral. Our

Chris Stasny (The STAZ) presides over STAZdom with help from his STAZsistant. He turns complex programming issues over to the STAZical Engineer and delegates number-crunching duties to the STAZtician. And let's not forget the legions of STAZlettes that oil the ever-grinding gears of the STAZomatic. This machine has turned out such titles as FutureBASIC II, Program Generator, The Menu Mill, and Classroom Publisher.

first mistake is to assume that the call somehow changes the order of resource files. This is impossible. Our second mistake is to assume that we can put things right by calling USERESFILE with the application's resource reference number. This is also a fantasy. Here is the only *safe* way to switch forks in mid stream:

```
origRes = FN CURRESFILE
CALL USERESFILE(resRef)

' Handle your file-specific resource operations here.

CALL USERESFILE(origRes)
```

Rules to Load By

Let's backtrack for a moment to re-examine the operations involved in opening a resource file. During this operation, the Resource Manager loads a map into memory that lists all of the blocks behind the doors of the newly opened file. If, during creation of that map, the Manager sees something that the programmer marked as "Pre-load," the information is actually moved from the disk into RAM. (The resource can be marked this way either by checking the appropriate box in ResEdit or Resourcerer, or by using SETRESATTRS.)

When we look for a resource with a statement like FN GETRESOURCE, we don't actually dig through RAM searching for our unique block. The Resource Manager scans the maps that are held in memory to see if an entry matches our needs. This entry must be the same resource type with the proper ID or name. Our Manager, clever fellow that he is, does not blindly load a resource when he locates the entry. He first checks to see if the resource has been previously loaded and, if so, returns a handle to the existing item.

Toolbox calls that obtain resource handles are governed by very specific rules. Normally, the resource is moved from disk to memory, but you can circumvent this in cases where you may wish to do something like record and list available resources. The following example records the names of all sound resources without actually loading them into memory.

```
DIM sndName$(100)

' Don't load - just look
CALL SETRESLOAD(_false)

' Get res count, but don't let it extend past
' the arbitrary limits of the string array.
theCount = FN COUNTRESOURCES(_"snd ")
if theCount > 100 then theCount = 100

' Get each resource handle and request
' info on it.
FOR i = 1 TO theCount
 rHnd1& = FN GET1INDRESOURCE(_"snd ",i)
 CALL GETRESINFO(rHnd1&,rID,rType&,rName$)
 sndName$(i) = rName$
NEXT

' Restore normal resource loading
CALL SETRESLOAD(_zTrue)
```

The example used COUNTRESOURCES and GETINDRESOURCE. These calls start at the top of the barrel and continue until they encounter the aging rusty steel at the bottom. (Don't forget, if USERESFILE was invoked, the top of the barrel may have been cut off, and our point of origin for the search may have started lower in the drum than expected.) We could have easily examined the resources that existed in one file to the exclusion of all others by switching to calls that contain "1" in their names. FN COUNT1RESOURCES would return the number of sound resources in the most recently designated file. FN GET1INDRESOURCE would retrieve the indexed resource only in that file.

Let's bring this all together in a routine that searches for sound resources in the System file.

```
DIM sndName$(100)
origRes = FN CURRESFILE

' Ignore anything above the System file.
CALL USERESFILE(0)
CALL SETRESLOAD(_false)

' We only want resources in the current file.
theCount = FN COUNT1RESOURCES(_"snd ")
if theCount > 100 then theCount = 100
FOR i = 1 TO theCount

' Get indexed resources only from the top file.
 rHnd1& = FN GET1INDRESOURCE(_"snd ",i)
 CALL GETRESINFO(rHnd1&,rID,rType&,rName$)
 sndName$(i) = rName$
NEXT
CALL SETRESLOAD(_zTrue)

' Put things back like we found them.
CALL USERESFILE(origRes)
```

We have now clearly established the limits of a resource operation. USERESFILE tells us where to start searching, and a "1" in the toolbox name tells us that we should stop after searching a single file. Absence of the "1" forces our search to extend to the deepest recesses of the drum.

Shared Responsibilities

In most cases resources are loaded into memory and used, rather than just being inventoried and listed. The conditions under which these items enter RAM is another perplexing methodology that begs for explanation. The complexity centers around the fact that the management of a resource becomes the shared responsibility of the Resource and Memory Managers.

The Resource Manager begins by examining a resource's attributes to see if it should be purgeable or locked, and allocates the correct memory through FN NEWHANDLE. It can even be loaded into the system heap instead of the application heap, so that it may be shared with other applications that can look downward from their side of the dividers and see it. You may recognize these attributes as check boxes in ResEdit dialogs that are labeled Purgeable, Lock, and System Heap.

Armed with attribute information, the Resource Manager asks the Memory Manager (politely, of course) for a block of memory that is purgeable (or not), locked (or not), or in the system or application heap. Information is moved into that block. This is (practically) the only time resource attributes are

examined — when the resource is loaded. From this point forward, the Memory Manager is in charge of the block. Memory Manager calls like HLOCK and HNOPURGE are used to handle the item as it floats around in the heap.

There is one very important exception to this rule. Never use CALL DISPOSEHANDLE on a resource handle. The Memory Manager would carry out its assignment by throwing away the information contained in the handle, but the Resource Manager (who was diligently tracking the block on his own personal tiny clipboard) would be left with a firm grip on an indeterminate chunk of RAM. It's not a pretty sight.

Lock and Purge

To lock or not to lock. That is the question. Most of the time, when I am called upon to debug a particular resource operation, I see abuses that would be too embarrassing to discuss on a daytime talk show. We have a tendency to ignore the attributes set up by the programmer who created the resource in the first place. As with SETRESLOAD and USERESFILE, we need to remain conscious of the creator's intent as we manipulate these items.

Assume that your application has a large preference resource that is loaded at startup. If we lock this piece of information in place, we have locked up a big, otherwise usable piece of binary real estate. Making it non-purgeable would have a similar effect. It may be our intent to move the entire resource into a global record or to examine individual pieces and branch to other operations. Either way, this resource would be marked as purgeable by its creator, but must temporarily be held in memory. We do it like this:

```
rHndl& = FN GETRESOURCE("PREF", myPrefID)
LONG IF rHndl&

' Record the status of the handle.
 hState = FN HGETSTATE(rHndl&)

' Make sure it stays in memory.
 OSErr = FN HNOPURGE(rHndl&)

' Handle resource operations here,
' then restore the handle to its original state.
 OSErr = FN HSETSTATE(rHndl&,hState)
END IF
```

Some toolbox routines handle purging without a programmer's assistance. Picture resources are among the most abused in all of Mac-dom. Take the following (really bad) example.

```
' Bad example! Don't try this at home.

DIM t,l,b,r

rHndl& = FN GETPICTURE(_myPictureID)
OSErr = FN HLOCK(rHndl&)
t:8 = [rHndl&]+_picFrame
CALL OFFSETRECT(t,-l, -t)
CALL DRAWPICTURE(rHndl&,t)
CALL DISPOSEHANDLE(rHndl&)
```

This example was so wretched that it was painful to type, but it represents the type of resource handling that is common in most programs. The first error was to lock the picture handle. If this picture was to be used often during the course of an application (as might be the case with a tool palette picture), its creator would have marked it as non-purgeable. In most cases, even the purgeable attribute would be turned off so that the picture could be expunged if required. Locking is generally a bad thing and would normally be done only on a temporary basis. (Remember HGETSTATE and HSETSTATE?) In our case, the HLOCK was unnecessary, since QuickDraw manages this particular resource during its operation, and undesirable since low-level picture operations may actually move or resize the picture resource.

Our next major mistake was to assume that the picture was loaded into memory. We immediately went to work on a resource handle without checking it for validity. Then, we disposed of the handle, which was being jointly tracked by the Resource Manager and the Memory Manager. Since DISPOSEHANDLE is a Memory Manager call, the Resource Manager was left out of the operation and is likely to cause serious damage when it next reaches for that resource.

Here's how we should have handled the operation.

```
DIM t,l,b,r

rHndl& = FN GETPICTURE(_myPictureID)
LONG IF rHndl&
 t:8 = [rHndl&]+_picFrame
 CALL OFFSETRECT(t,-l, -t)
 CALL DRAWPICTURE(rHndl&,t)
END IF
```

If this was a startup picture, it would probably never be needed again. The moment that the memory it occupied was required for some other operation, the picture would be purged from RAM. On the other hand, if the application's partition was large and the user's requirements small, it might remain in place for the duration.

Later, when the user decided to view the application's "About" box, we would use exactly the same routine to display the picture a second time. If plenty of space was available and the picture had managed to remain in memory, it would not be reloaded. A handle to the existing data would be returned from FN GETPICTURE. If constraints had forced the resource out of memory, it would be reloaded as a result of the call. This debugged example was more memory-efficient, and took fewer lines of code.

Handle Hand-off

While the ownership of resource handles seems to be preordained, we may still exercise control over who will ultimately own the handle. A call to DETACHRESOURCE does this. It pulls the resource's entry from a file's resource map. It does not remove the handle from memory or change the attributes of that handle. After this call, the Memory Manager controls the block, and the Resource Manager ceases to acknowledge its existence. If you wish to dispose of it, you'll have to use DISPOSEHANDLE.

While there is no call that completely releases a Memory Manager handle to the control of the Resource Manager, there is a procedure that turns a handle into a resource:

StoneTable Break on through the limitations of the List Manager

	Name	Home Address
Father	**John Smith**	3224 NE 58th Avenue P.O Box 12665 Portland, OR 97212

68 & PPC libraries for Think C, CodeWarrior C & Pascal, MPW C & Pascal $200.00

Includes PowerPlant Class for CodeWarrior
International shipping (US Airmail) $10
No royalty fees for applications

StoneTablet Publishing
P.O. Box 12665
Portland, OR 97212
voice/fax (503) 287-3424
stack@teleport.com

More Info - http://www.teleport.com/~stack
Demo - ftp://ftp.teleport.com/pub/vendors/stack/StoneTableDemo.hqx

Row and column titles
variable size row and columns
move, copy, sort, hide, resize rows and columns
set font, size, fore/back color, face, alignment per cell
cell margins, top/bottom & left/right
drag cells in and between tables
edit cells in place
popup menus & check boxes
draw boxes around multiple cells
variable size grid lines
PICT's and controls in cells
"LDEF-like" custom drawing function
greater than 32K data per table
plus all List Manager functions and more

New Price
New Version

ADDRESOURCE. This call takes a great deal of setup and is best documented in FN pGreplaceRes in the Runtime.INCL file of a Program Generator project. But there are some important rules that deserve attention. When a resource is added, it is added to the current resource file. Use CURRESFILE and USERESFILE to change this during such operations. Adding a resource is a blind operation that allows duplication. If you add a preference resource to your application three times, you'll end up with three copies of the resource in your file.

Slow Execution

Because most folks have a hard time understanding resources, there is a tendency to write out a resource every time it is modified. Apple engineers are no exception. Most of us remember the early days of the Macintosh 840AV. An uninformed ROM rat patched the toolbox call CHANGEDRESOURCE to make it call UPDATERESFILE each time it was called. This was a bad thing. Let me explain why.

When your program calls UPDATERESFILE or WRITERESOURCE, the entire file may be written to disk. Remember that resources are not really magical. They occupy space on a disk, and are ordered according to strict internal guidelines. If you replace a picture that is 50K with one that is 100K, there is no sorcerer's potion that compresses the new resource into the same space used by the old one. The file is simply rewritten, starting at the beginning of the old 50K resource.

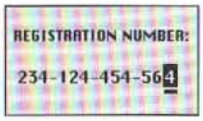
When the AV's were first released, folks complained that resource handlers like ResEdit and Program Generator slowed to a crawl. This is because a simple operation (CHANGEDRESOURCE), which was supposed to set a single bit in a single flag, ended up rewriting the file. The reality is that it is almost never necessary to update the file or write its resources. This happens automatically when the file is closed.

There is one additional feature of CHANGEDRESOURCE that deserves attention: You should never call CHANGEDRESOURCE on a purgeable block. When the file is closed, the Resource Manager scans its list to see what has been modified. Then it rewrites the file with these new items in place. If, in the interim, the Memory Manager has purged the block, you will see random data written to the disk. If you have ever seen a file explode from a few hundred K to several megabytes, this is the likely reason. Mark resources in the following manner to prevent this problem.

```
OSErr = FN HNOPURGE(rHnd1&)
CALL CHANGEDRESOURCE(rHnd1&)
```

Conclusion

Our list of rules to live by grows ever more complex. In the olden days, we were just supposed to remember not to tug on Superman's cape or spit into the wind. Now we must watch resource attributes and referee a tenuous cease-fire between two headstrong Macintosh Managers. My friends from Trinidad might beg for simplicity. "And me anudder monager. Dees ones too complex." MT



By Jamie McCornack

Serial Killer

Easy and effective serial number management for electronic game distributors

This material isn't rocket science — it is just an easy way to do something that you need to do. The computer game business is a dog and pony show, and this is about how to deal with pony poop. I am offering no glamour here, but I might save you a couple of day's work.

An Important Part of Every Breakfast

Games need serial numbers more than any other software. They also need individual serial numbers for individual customers because

- People play games for fun, and paying for things isn't much fun. Given a choice, the average game player will say, "It's only a game," and never think twice about ethics. These people would never go to their Porsche dealer and take off with a red one, saying, "It's only a Porsche," even though Porsches are supposed to be fun too.
- People will respond positively to paying for games if they are reminded gently and given the opportunity to pay at their own pace. Ambrosia recently bought a company Humm-V from voluntary contributions — folks who were willing to send in $15 to $20 for a serial number.
- The reward for having a serial number need not be big, but there must be some reward. Ambrosia (to use the most successful example in the Mac electronic distribution world) offers contests and the like to registered users. However, the main reward is that the game stops asking for a serial number every time you play it. While the reward need not be big, it needs to be immediate. So, the game itself needs to respond to serial number input.
- Having a single serial number hard-wired into your program doesn't work. It will immediately find its way onto the Internet, and the folks who "crack" your program will think that they belong in the next William Gibson novel.

The glossy box people do not have this problem as much, since they publish mostly on CD-ROM which costs peanuts to reproduce, but much more to copy. To copy protect a CD-ROM, all you have to do is insist that the CD be present (or some 200meg file on the CD) before the program runs. At the current state of the art, your CD costs you under a buck to produce, and a copy costs the freelance pirates about $10 — so, they might as well buy an original.

However, if distributing games on floppies, compilation CDs, or via modem, the economics of scale work against you.

A floppy loaded with your data and labeled costs more to produce than a blank floppy (purchased in boxes of 20) costs a potential pirate. Imagine what the book business would be like if books cost publishers six cents a page to print, and every person in the country had a nickel-per-page copy machine at home.

Compilation CD's (One Thousand Great Games for $19.95) are an effective way spread around demos and shareware games. However, the buyer has my game (and 999 other games), the distributor has $19.95, and I have nothing. Now I need to encourage the buyer to spend a little more. Electronic

Jamie McCornack is an amateur programmer and professional writer in the Mac game genre. He is a project editor for Hayden's *Tricks of the Mac Game Programming Gurus* and he wrote *Real Life Gamewerks* for Metrowerks' Gamewerks SDK.

distribution is the same scenario, except the buyer receives a bill from AOL at the end of the month, and I still have nothing.

Well, not quite nothing. What I have is a marketing opportunity. The opportunity to sell something — an end to the "please register" reminder, a game enhancement, and early access to my next game. The easiest and most effective item for this market is a serial number. Here are some serial number guidelines.

- The serial number should be unique, and the gratification should be immediate. The game program needs to accept and reward any valid serial number, and spurn any invalid serial number.
- There should be more invalid numbers possible than valid numbers. There should be a minimum of 1000 wrong numbers for every right number or the "Hunt and Peck Hackers" will perform brute force solutions instead of doing their history homework.
- The system should be easy for you, easy for the registered user, and easy for the folks that answer your 800 line, fill out the MasterCard slips, and give out the serial numbers.

What does Generator Generate?

Generator cranks out a couple thousand serial numbers per second, and saves them as a tab delineated Excel file. Most databases can import Excel data. You can select start and end numbers from 1 to 999999. I recommend that you generate a reasonable quantity of numbers — you probably don't need a million serial numbers yet, nor do you want to devote 25meg to storing the file. Note that Generator is a 68k Mac program, but since it produces about 120,000 serial numbers per minute, it is probably not worth writing an Accelerated for PowerMac version.

I import my serial numbers into a FileMaker Pro document. FileMaker garbles the first record in the list because it stumbles on the Excel header, so I start my list with 0. I delete the first record, which leaves me with serial number 1 heading the list.

Generator's first function, CalculateValues(), names the file, sets the Excel header, forces tabs into the data where needed, and saves the serial numbers and customer numbers.

Making the numbers unique

For clarity, the twelve digit numbers generated by Generator are grouped in four clumps of three; xxx-xxx-xxx-xxx. They look just like Bonkhead's numbers, but they're not. The uniquifying algorithms used herein are different from theirs, and yours should be different too. You'll have to write your own unique ProduceValues() function if you want your own unique serial numbers, but this version of ProduceValues() will give you some ideas.

Idea #1: ProduceValues() does much of its work with strings, with numerical input converted via LongToStr(). With minor changes, your version can output numbers with characters (both upper and lowercase) and symbols as well as digits (yes, the results are still called serial numbers).

Idea #2: LongToStr() places a string length value in str[0], and a nul value in str[last char + 1], creating a string that is both a Pascal string and a C string. Why? Someday you might be programming cross-platform, and the Mac prefers Pascal strings, and Wintel prefers C strings, and I don't know what Nintendo or the PlayStation use. However, there are a lot of potential customers out there, and they don't all use Macs. Sure, it costs us a character, but who is going to key in a 255 character serial number?

ProduceValues() briefly converts the string values '0' through '9' back to single-digit numbers, this is purely for the sake of clarity. For example, if you want to use uppercase letters instead of digits, convert

```
d1 = valStr[i++] - '0';
```

to

```
uL1 = valStr[i++] - 'A';
```

and the number-stirring line (which converts 123 to 321, in this example), from

```
val1 = (d3*100)+(d2)+(d1/100);
```

to

```
val1 = (uL3*676)+(uL2)+(uL1/676);
```

This converts ABC to CBA. Using uppercase letters makes the serial "number" digital in base 26, giving 676 possible two letter combinations. Using uppercase and lowercase letters, plus the ten numerical digits, gives a base 62, with 3844 two character combinations and about 15 billion four character combinations. No wonder Apple hasn't run out of owner resource ID numbers yet.

In this example, the first and fourth three digit number groups reflect the customer number; 000-xxx-xxx-001 is customer #1, 076-xxx-xxx-345 is customer number 76,345, etc. The second number group is a random-appearing response to number group one, and the third number group is a random-appearing response to number group four. So, the first 999 customers are customers #1 through #999...

Why don't we have a customer #0? *We* are #0. #0 is for in-house testing.

Customers #1 through #999 all have the same second number group (930, in this example) and customers #1 and #4,001 and #38,001 all have the same third number group (it happens to be 228).

The algorithm for generating response numbers is pretty simple in this example.

- Get a three digit number group (e.g. 123).
- Add 13 (e.g. 136).
- Swap the first and third digits (e.g. 631).
- Multiply by 3 (e.g. 1893).
- Subtract the original * 2 (e.g. 1893 - 246 = 1647).
- If the result is negative, convert it to its absolute value.
- Discard all but the last three digits (e.g. 647).

In real life you will want to use different seed numbers to

add (or subtract) in step 2, different swap patterns, different multipliers, and you may want to swap and/or mingle digits from the first and fourth number groups. There are many ways to make your particular serial numbers your own.

Simon says...play

The game program requests serial number input from the player, and confirms or denies the validity of that number. Does it run Generator backwards? Nope.

The game generates second and third number groups in response to the first and fourth number groups using exactly the same code as Generator, and checks them against the second and third number groups entered by the player. Are the generated and keystroked numbers identical?

If you are a believer in the heavy-handed school of registration numbers, the game can refuse to run until a suitable number is entered. A lot of glossy-box software uses this system. The only value I see from it is that if the buyer sends in a registration card with the number thereon, you know who to yell at if that number shows up on a BBS or on the pirates.com home page (my apologies if there really is a pirates.com out there).

If you are light-handed, a simple "thank you" splash screen might well suffice, and deactivate the gentle registration reminders that keep popping up. Also, flag this copy of the game as registered, either in the prefs file or in the application itself — do not demand the number be keyed in before every game.

The Number of the Beast

Simon Beeblebrox, #958331

One thing electronic distributors can do that glossy box distributors cannot, is personalize serial numbers. If someone buys a game off the shelf, they expect to be able to go straight home and play it with no further hassle. However, a customer who is mailing in a registration fee for the enhanced version of an electronically distributed game, or is phoning you with a pencil in one hand and a VISA card in the other, is going to give you their name. If you like, you can integrate that name into their serial number.

NameNumerator converts the first 24 characters of a name to a six digit decimal number. The six digit number (e.g. 958331) could be combined with the previous serial number generator (e.g. 958-xxx-xxx-331) to create a real challenge for the crackers.

Let's take a look at NameNumerator's most significant function.

Get Value

GetValue() takes a string up to 255 chars long, and returns a six digit decimal number, in the form of a six character string.

```
void GetValue(Str255 valStr, Str255 retStr)
{
  long i;

  for (i=1; i <= 6; i++)        //Load first six char codes
    retStr[i] = (valStr[i] % 10);
  for (i=1; i <= 6; i++)        //Load next six char codes(7 through 12)
    retStr[i] = ((retStr[i] + valStr[i+6]) % 10);
  for (i=1; i <= 6; i++)        //Load third six char codes (13 through 18)
    retStr[i] = ((retStr[i] + valStr[i+12]) % 10);
  for (i=1; i <= 6; i++)        //Load fourth six char codes (19 through 24)
    retStr[i] = ((retStr[i] + valStr[i+18]) % 10);

                                //Add 257458 (kinda...) and '0'
  retStr[1] = ((retStr[1] + 2) % 10) + '0';
  retStr[2] = ((retStr[2] + 5) % 10) + '0';
  retStr[3] = ((retStr[3] + 7) % 10) + '0';
  retStr[4] = ((retStr[4] + 4) % 10) + '0';
  retStr[5] = ((retStr[5] + 5) % 10) + '0';
  retStr[6] = ((retStr[6] + 8) % 10) + '0';
}
```

The first half of GetValue() takes the individual characters of a Str255 (the registrant's name), strips all but the last digit of that character's ASCII code, and places that digit in another Str255. After the first six digits are placed (valStr [1] placed in retStr[1], valStr [2] placed in retStr[2], etc.), the next six are added to the first six (valStr [7] added to retStr[1], valStr [8] added to retStr[2] etc.) and all but the last digit is stripped from the resultant. This process is repeated through 24 characters. However, to distinguish Clifford Hummel Throckmorton-Whitney from Clifford Hummel Throckmorton-Whitfield, you can repeat as necessary.

The second half of GetValue() adds the arbitrary seed number 257458, sorta...

Well, it's not quite addition. Actually, we are adding the digit from the hundred thousands column to retStr[1] and stripping off all but the least significant digit from the result, adding the digit from the ten thousands column to retStr[2], and so on. Then we add the ASCII value of the character '0', so instead of having the actual digit values in the string, we get the ASCII values of the digits. 0 + '0' = '0'; that is, 0 + 48 = 48, which is the ASCII code for the character '0'. 1 + '0' = '1'; that is, 1 + 48 = 49, which is the ASCII code for the character '1'.

So you have your telemarketing people with NameNumerator up and running on their Macs, and when folks call in their orders, they can say, "How do you spell your name, Mr. McCornack? Your serial number is..." and that is why you need to use genuinely arbitrary seed numbers.

For example,

```
for (i=1; i <= 6; i++)                  //Load first six char codes
  retStr[i] = (valStr[i] % 13);         //In base 13
```

will throw a Spaniard in the works, and

```
for (i=1; i <= 6; i++)                  //Load first six char codes
  retStr[i] = (valStr[i] % (10 + i)); //In base (10 + i)
```

will scramble things even further.

Or perhaps you'd like to use alphabet characters. To start the above six digit example with an uppercase letter, change the first line of the second half of the routine to

```
retStr[1] = ((retStr[1] + 2) % 26) + 'A';
```

which adds a random-like number from 0 and 25 to the ASCII code

for 'A', giving the full range of capital letters (since 'A' + 25 = 'Z').

Conclusion

Big Brother, #964752, is Watching

What is the advantage of personalized serial numbers? Many people who would otherwise distribute a generic serial number will balk at distributing a serial number which only works with their name, thus exposing them as the culprit. However, the typical game pirate is not worth harassing — what are you going to do, have your lawyer's mom call the pirate's mom to tell her she has a naughty kid?

You could build a serial number from a name and phone number, and verify the phone number by having your telemarketers say, "We will call you back in three minutes with your serial number" when the order is placed. However, if I were publishing a high-bucks 3D animation program, I would want to discourage folks from buying one copy and putting it on every workstation in the art department. I could say, "You don't like dongles? Then I will give you a serial number. However, it will only work in conjunction with your name, your credit card number, and its expiration date. If your system isn't secure enough for your credit card data, maybe it isn't secure enough for our program, either. Are you sure you don't want a dongle?" MT

Help Make MacTech Work

Here at *MacTech Magazine*, we rely heavily on outside writers for most of the material that appears in our pages. If readers did not participate in the magazine, sending us their ideas and taking the time to write articles, there would be no *MacTech*. *MacTech Magazine* is not a staff of writers sending a constant stream of one-way messages outwards; it's a living, evolving network of readers conversing with one another, educating one another, sharing their knowledge, their experience, their interest, their trials and tribulations and joys and successes in the constantly unfolding story of programming the Macintosh. *MacTech Magazine* doesn't just happen: it's what the community makes it. If we carry reports of future trends and technologies, if we teach useful methods, if we review new books and tools, if we provoke thought, provide help, ride the wave of current interests and concerns, it is only because we reflect the thoughts of our readers, who speak through our pages.

You are invited to involve yourself in this exciting conversation amongst readers. No matter who you are, no matter what your credentials may be, if you have a tale to tell, a trick to share, a technique to teach, we want you to consider joining the family of those who write for *MacTech*.

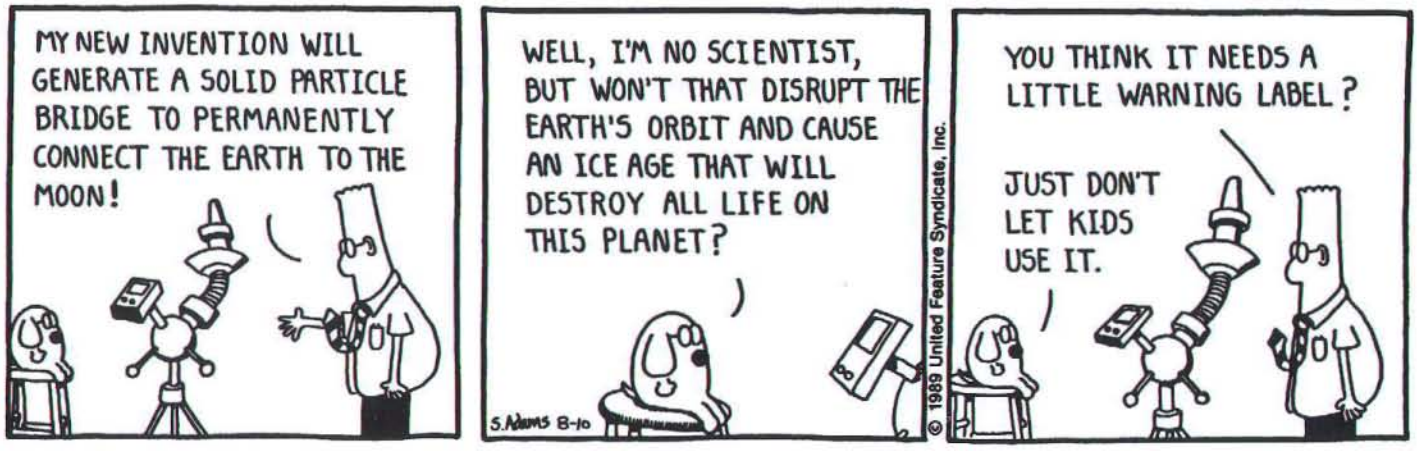
Don't just wait for a topic to be covered or a technique explained in *MacTech!* Take responsibility! Write us an article yourself!

To write for *MacTech*, just send for our Writer's Kit. It's a Microsoft Word file containing the Styles you need to use, and giving lots of helpful advice and information, including all the legal stuff. You can let us know what you're writing about, or, if you want to, you can just write the article and spring it on us when it's done. [**Note:** We also have a need for people willing to make themselves available to write occasional product/book reviews.] If we publish your article, you'll be paid for it!

Write to us, the editorial staff, at editorial@mactech.com (or one of the other addresses listed on page 2 of the magazine). Take the future of *MacTech Magazine* into your own hands!

For Macintosh Programmers & Developers
MacTech™ MAGAZINE

Dilbert® *by Scott Adams*



By Dave Mark, *©1996 by Metrowerks, Inc., all rights reserved.*

Mike Lockwood, Debugger

One of the most important, yet least understood part of the Mac development process is the debugger. Sure, everyone knows how to use the debugger to step through your source code, but I've always wanted a peek beneath the hood, a chance to see how a debugger really works. This month's interview is with Mike Lockwood, debugger extraordinaire.

Dave: You worked at Apple four years before you came to Metrowerks. What did you do there?

Mike: I first started working on development tools on my own, before I joined Apple. I wrote my own MacApp debugger as a side project, and when I got it working I sold it to Apple. Apple contracted me to finish it, and it shipped on E.T.O. as SourceBug. Soon after that, I joined Apple as an employee and worked for a year on the debugger for a new integrated development environment that was to replace MPW.

After that project was canceled, I transferred to the System Software group and worked on the Finder team. At the time there was no source level debugger that could debug Finder extensions, so I kept working on the debugger from my previous project — so I could use it myself. I added support for debugging threads and shared libraries, and it was soon being used by a number of different groups within Apple. Since it was an underground, guerrilla programming effort, I decided to call it VoodooMonkey. Then when no one was looking, someone snuck it on one of the Developer CDs and VoodooMonkey shipped outside of the company. Since then, it became somewhat of a cult classic.

After working on the Finder team for awhile, I became interested in working on developer tools again and wanted to move back to the East coast. So I transferred to the Cambridge office to work on the Dylan project. There, I wrote the application framework for Apple Dylan, and served as the main guinea pig for the Dylan compiler and development environment. After that project got canceled, there was an opportunity for me at Metrowerks, and I joined them the day after I left Apple.

Dave: How did you hook up with Metrowerks?

Mike: Metrowerks first contacted me in the Fall of 1993. They needed a 68K debugger nub for CodeWarrior, Apple only had a PowerPC version of DebugServices up and running. Greg Galanos called me to see if he could use DebuggerINIT, which was the debugger nub I wrote for VoodooMonkey. At the time, I didn't know much at all about Metrowerks, other than they were working on a C/C++ compiler for PowerPC. I wanted to help them out, but I also wanted to avoid some of the political problems I ran into after VoodooMonkey shipped. So, I directed Greg to go through the proper channels in Cupertino.

I knew Greg would never be able to cut through all of the red tape, and I wasn't sure I would ever hear from him again. However, he called me two days later and said "Hey, I just licensed the sources to DebuggerINIT from Apple." Now I knew I was dealing with a force to be reckoned with.

I continued to talk to Greg and other people at Metrowerks, and provided some informal tech support for DebuggerINIT. When it was clear that the Dylan project was going to be shut down, I talked more seriously with Metrowerks about joining the company. Dan Podwall, the original author of the Metrowerks debugger, was busy working on the browser and they needed someone to take over the debugger.

Dave: What exactly is a debugger nub?

Mike: A debugger nub is a small program that controls a program that is being debugged, and is responsible for things like setting breakpoints and single stepping through code. It provides a layer between the main debugger application and the low level system stuff. The distinction is important in the case of two machine debugging, because the debugger nub runs on the machine being debugged and the debugger application runs on a different machine. MetroNub is our debugger nub for single machine MacOS and Java debugging. We have other debugger nubs for debugging code running on other platforms.

Dave: How does MetroNub interface with the Metrowerks debugger?

Mike: MetroNub is a system extension that provides an API to the Metrowerks debugger that supports debugging functionality like single stepping. MetroNub exports a structure full of mixed mode function pointers via a Gestalt selector that the debugger application can call. MetroNub also interfaces with various parts of the OS. For example, it patches the Resource Manager to track where code resources are loaded in memory, and uses an undocumented Process Manager API to suspend and resume a process that is being debugged. It also patches the 68K exception vectors and registers callback functions with the PowerPC Exception Manager and the MP kernel so it can be notified when exceptions occur.

Now let's examine what happens when you single step through a line of code. At this point, the program is already stopped, meaning that MetroNub had already told the Process Manager to suspend the program's process and make it ineligible for scheduling. The debugger application then calls the MetroNub "Step" command. When it makes this call, it would specify to MetroNub whether to step into or step over function calls. It would also specify a range of addresses for the program counter so MetroNub will not stop execution again until the PC is outside of this range. When stepping though source code, this range would correspond to a single line in the source code.

MetroNub would record the PC range specified, and would then tell the Process Manager to resume the process. The process is then eligible to be scheduled again, but will not actually resume until the debugger application calls WaitNextEvent(). After it is rescheduled, some of MetroNub's code that was left on the call chain of the program being debugged will resume execution. Before MetroNub returns control to the program being debugged it restores all the registers and sets the trace bit in the processor's status register. The trace bit tells the processor to cause a trace exception after each instruction is executed. MetroNub receives these trace exceptions, and processes them until the program counter falls out of the specified range. Then MetroNub tells the Process Manager to do an immediate context switch from the program being debugged to the debugger application. This leaves some of MetroNub's exception handling code on the call chain of the program being debugged, which will resume after the program being debugged is rescheduled by the Process Manager.

Dave: What language is MetroNub written in?

Mike: MetroNub is written almost entirely in C++, with just a small amount of 68K assembler. The assembler is used only in a few places where it is necessary to access registers directly or execute some uncommon 68K instructions. For example, the 68K exception vector patches and some of the Resource Manager patches are written in assembler. MetroNub is mostly 68K code, and only contains a small amount of PowerPC glue code for some of the callbacks it registers with the OS.

Dave: But how does the link between the source code MetroNub happen?

Mike: MetroNub doesn't know anything about the source code. The debugger application handles the mapping from source code to object code offsets, based on information in the sym file, which is generated by the compiler and linker. Suppose you want to set a breakpoint at a certain line of source code. The sym file contains a table that maps the offset in the source file to an offset in the object code for the function containing that line of code. A different table in the sym file is then used to translate the offset in the function's object code to an offset into a CFM code fragment or a code resource. The debugger then tells MetroNub to set a breakpoint at the given offset within the code fragment or code resource.

The debugger application does not have to know where the code is loaded in memory or if it is even in memory at the time the user sets the breakpoint. If it is loaded, the debugger will convert the offset to an absolute address and insert a trap instruction at the address where the breakpoint should be set. If the code is not loaded yet, MetroNub remembers the breakpoint and installs the trap instruction after the code is loaded but before the program is able to execute the code.

Dave: What changes did you have to make to your design to deal with Java?

Mike: Fortunately, no major design changes were needed to support Java. That was good news for us, because time-to-market was important. Our debugger could already debug two different processor architectures (68K and PowerPC) simultaneously, so adding a third architecture was not difficult. We just treated Java byte codes as if they were assembler instructions and the Java VM as if it was a microprocessor, and the debugger architecture we already had in place worked fine. I had to write a new implementation of the symbolics code so it could read symbolic information from Java class

files and Zip files instead of a sym file. I also had to add an API to the Metrowerks Java VM so MetroNub could communicate with the Java VM and control the execution of Java code. This approach had the nice side effect that it is possible to debug both 68K and PowerPC C/C++ code and Java code simultaneously with a single debugger.

Dave: A lot of companies have said that they will no longer support the 68K. Do you think learning to debug 68K code is a worthwhile investment?

Mike: 68K is becoming less important, but a good understanding of 68K code is valuable when debugging on System 7, even if you are writing PowerPC native code. Since much of the operating system is still emulated 68K code, it is not uncommon to have to debug 68K code on a PowerMac.

Dave: What's your feeling on how people should approach debugging? Should they learn MacsBug? Should they buy a copy of Jasik's Debugger? Learn PowerPC assembler? Got any good book recommendations? Are there utilities people should definitely have on their hard drive?

Mike: The Metrowerks debugger is strictly a high level debugger. You also need to have a low level debugger like MacsBug or Jasik's debugger installed when debugging with the Metrowerks debugger. Jasik's debugger is actually both a low level and high level debugger, so if you like Jasik's debugger, you don't even need a copy of the Metrowerks debugger! Jasik's debugger is very powerful and has a lot of features that no other debugger has. But many people find the Metrowerks debugger easier to use.

Nowadays, it isn't critical that you learn PowerPC assembler, unless you need to write very low level or very optimized code. But for debugging purposes, it is often useful to have basic reading knowledge of assembler, even if you do not write it yourself (I have written a lot of 68K assembler over the years, but haven't actually had a need to write any PowerPC assembler yet). But understanding how to read and debug at the assembler level can be very useful at times.

There are a number of utilities available that can help you with debugging. You can use the Debugging Memory Manager from Apple to help catch memory problems in your application heap. Similarly, you can use the DebugNew library that comes with CodeWarrior to track down memory problems in the built-in C/C++ memory allocater. Zone Ranger, which also comes with CodeWarrior, can be helpful monitoring memory usage and watching for memory leaks. There are a number of extensions, like EvenBetterBusError, DoubleDispose, etc. that are available from Apple that watch out for common programming errors when using the Memory Manager or Resource Manager. Onyx Technology has a tool called QC that monitors all Memory Manager calls and checks for these programming errors. A demo version of QC is available on the CodeWarrior Reference CD. MT

There are a number of extensions, like EvenBetterBusError, DoubleDispose, etc. that are available from Apple that watch out for common programming errors when using the Memory Manager or Resource Manager.

Want to subscribe to MacTech Magazine? Contact <mailto:orders@devdepot.com> call 800-MACDEV-1 or 805-494-9797, or fax 805-494-9798.

Mike Lockwood is the debugger architect at Metrowerks. He is responsible for the overall debugger architecture as well as debugging support for the Macintosh and Java. Prior to joining Metrowerks, Mike worked on debuggers and other development tools at Apple Computer.

SYMANTEC TOP TEN

By Kevin Kenan

SYMANTEC.™

This month's Top Ten ranges over several Symantec products, but we begin with a couple of issues regarding Visual Page, Symantec's new WYSIWYG HTML editor

Visual Page is an excellent companion to Visual Café, and a preview release can also be downloaded from <http://cafe.symantec.com/>. We follow the Visual Page section with questions involving Café and its documentation generator and finally we address a problem with the Visual Architect.

Q: How do I use anchors in Visual Page?

A: In order for the Anchor icon to become enabled, your Visual Page document must first be saved to disk. Then, select the text that will serve as the anchor, and click on the Anchor icon or choose Anchor from the Insert menu. A small anchor marker will appear on the left of the text that was selected. Double clicking the marker will bring up a dialog box for naming the anchor. A mouse click on OK saves the name.

To generate a link to the newly created anchor, select the text that you want to serve as the colored link and simply drag the anchor marker and drop it on the selection. This even works between Visual Page documents.

Q: How do I create a link to another page (as opposed to an anchor in another page) using Visual Page?

A: There are several ways to link to another page and the method you use depends on where the page is and how well you know the URL. If you know the URL you can simply select some text and type the URL in the "Link To:" field at the bottom of Visual Page. After you press return, the selected text is linked to the URL. There are, however, easier and more elegant ways to manage your links.

To use drag-and-drop to link files, simply drag an HTML file from the Finder and drop it onto some selected text. The selected text is now linked to that HTML file. It is easier to update the structure of your links if all your HTML files are grouped together in a single folder, but if you decide to organize your pages in a set of nested folders, you must be cautious when altering the structure of your nested folders or else your links might break.

When you want to link to a page on the Internet, Netscape's drag-and-drop capabilities make this otherwise laborious cut and paste operation quick and intuitive

When you want to link to a page on the Internet, Netscape's drag-and-drop capabilities make this otherwise laborious cut and paste operation quick and intuitive. Using Netscape, navigate to a page that has links you want to duplicate. Drag a colored link from the Netscape window and drop it onto some selected text in Visual Page. The selected text and that web page are now linked. Note that you can also drop the text from Netscape anywhere on your Visual Page document, not just on selected text. In this case, Visual Page simply creates a link using whatever text identified the link in Netscape.

Q: How do I embed a Java applet into my Visual Page document?

A: For an example, we will use the Animation demo which you can find in the Demo Projects:Intro to Java

Finish your installer *in record time* with DragInstall®

You've run a long race.

You're almost done with your product.

Now that it's time to build your installer, don't let complex licensing arrangements and complicated tools trip you up.

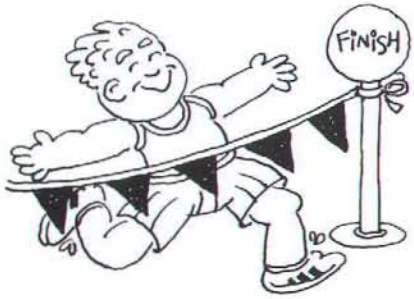


Download a demo of DragInstall
from http://www.sauers.com/draginstall

Build your installer
in a single step with DragInstall's QuickScript feature

Register DragInstall
with a single phone call to 1-800-890-9880

And cross the finish line
total time: 1 hour, total cost: $300



For more information:

Ray Sauers Associates, 1187 Main Avenue, Suite 1B, Clifton, NJ 07011 USA
voice: 201.478.1970, fax: 201.478.1513, email: info@sauers.com

Programming:Animation folder on your Café CD. If you installed the default distribution of Café, then the Demo Projects folder will also be on your hard drive. First generate the .class files by selecting Build Applet Classes from the Build menu. When the compiler is done, we can embed the applet in a Visual Page document.

1) Open the Animation folder in the Finder; you will see that Café has added two new files to this folder: Animation.class and Animator.class.
2) Bring your Visual Page document to the front so that you can still see the icons for the new .class files.
3) Drag the Animation.class file onto your Visual Page document and drop it wherever you please. A dotted box appears with a picture of Duke waving and the label Animation.class.
4) Double click on the picture of Duke and a dialog box appears allowing you to set the Applets parameters.

You can now explore the Java API in the comfort of your own web browser.

---

5) Change both the Width and Height to 160 and click OK.
6) Save your Visual Page document and you are ready to view your new, applet-enhanced web page in any Java aware browser.
7) Choose Browse in the Edit menu to open your Visual Page document in a web browser. You may first need to tell Visual Page where your browser is by selecting Preferences... in the Edit menu and clicking on the browser icon.

We dragged Animation.class rather than Animator.class onto the Visual Page window because the Animation class extends the Applet class. In general, always drop the .class file that extends the Applet class.

Q: Does Café implement Java's documentation generator?

A: Yes. In fact, included in your Café distribution you will find HTML documentation of the entire Java API. They are located at Symantec Café for Macintosh:(Java Libraries):API Documentation. Double click on the packages.html file to open it in Netscape. You can now explore the Java API in the comfort of your own web browser. Integrating the documentation for your own classes into this collection simple requires a menu selection.

1) Open a project window — the Animation project we used in the previous Answer works just fine.
2) Select a .java source file.

Distribute Your Software Electronically and Get Paid!

PACE Software authorization systems let you:

- Create fully functional time- or launch-limited "trialware" without changing your source code
- Distribute via the Internet, CD-ROM, BBS, floppies and more
- Unlock software with flexibility and security
- Add registration-capture to your software without writing code
- Integrate registration and authorization with popular web servers



P A C E
ANTI-PIRACY

To sample the customer experience, download our demo from www.paceap.com. It's easy, and it's free!

Want to try out our system on your software? Call 408/297-7444 or send email to sales@paceap.com to order our evaluation kit.

1082 Glen Echo Ave
San Jose, CA 95125
vox: (408) 297-7444
fax: (408) 297-7441
email: info@paceap.com
web: www.paceap.com

3) Choose Produce Documentation from the Build menu.
After a few seconds a documentation file for each class in the .java source will appear in the API Documentation folder. If you used the Animation project, then two new files, Animation.html and Animator.html, were produced. Generating documentation for an entire project is simple as well.
4) Bring you project's window to the front.
5) Select Generate All Documentation from the menu.
A dialog box will ask you to confirm your desire to produce the documentation; press OK and off you go.

Q: I followed the above directions with the Animation project, but no documentation was produced. Why?

A: The documentation files were most likely produced, but placed in a different folder. You can specify where a project's documentation is produced by selecting Options... from the Project menu. In the window that opens click on the Project Type icon in the left column. You can now set the Documentation directory as you choose. Factory Settings direct the documentation to the API Documentation folder.

Q: I set the Options so that my project's documentation is produced in the project file, but now when I browse the HTML files in Netscape the graphics do not appear and links to other java classes are broken.

A: The project manager expects all documentation files to be together in one folder though you can specify which folder. A good strategy for keeping all your documentation in a single folder while retaining access to them from your project file requires you to first generate the documentation in the project folder. Then make aliases of the files, and move the originals to the API Documentation folder or whatever folder you have specified for your documentation. Now when you double click on one of the aliases, Netscape opens the original and the graphics and links work.

Q: I like having the documentation produced in my project folder, but I would rather not reset the options for each new project. Is there a way to tell the project manager to always set the appropriate options whenever I create a new applet?

A: Most definitely. Inside Symantec Café for Macintosh:(Project Models):Java Applet you will find a project file called @1. Open this project by dragging it onto Café's project manager. Set the options for this project as you wish and close the project. Every new applet you create will inherit those options. You can even go so far as including a folder inside the Java Applet folder and set the options to always produce the documentation in that folder.

Q: When I use the Class Browser in Café, and double click on an italicized class name, I get an error of -43; I would rather get a description of the class.

The Class Browser tells your web browser to look in whatever folder you have specified for documentation. If you are producing documentation in your project file, the web browser will not find the java classes

A: An italicized class name means that the java source for that class in unavailable for the Class Browser to read. Instead, you can set up InternetConfig (included in the Third Party Goodies folder on you CD) to point to your web browser. Then, when you double click the class name your web browser opens to display the HTML documentation as described earlier. An important detail to keep in mind, though, is that the Class Browser tells your web browser to look in whatever folder you have specified for documentation. If you are producing documentation in your project file, the web browser will not find the java classes.

Q: I am experimenting with packages by putting the above Animation project into a package called my_java_classes.my_animation. But when I try to generate the documentation all I get is the HTML file for the Animator class. What happened to the documentation for the Animation class?

A: Documentation files are named after the full package name, and when your package name is prepended to the class names, you end up with a file name longer than the Macintosh's maximum of 31 characters. The overflow characters are simply cut off and in this case the resulting names are identical. Both documentation files are produced, but the last one produced overwrites the first. To resolve this, use shorter package names.

Q: Visual Architect v8.1 does not seem to create a floating pallet with the drag bar on the side rather than the top. Why not?

A: Visual Architect v8.1 apparently generates a faulty WDEF for floating pallets. A resource internal to VA is used to generate the WDEF, and the solution requires us to replace this internal faulty resource.

1) Retrieve the Infinity Windoid WDEF 2.6 code resource from <gopher://micros.hensa.ac.uk:70/40/micros/mac/finder//m/m036/>. The file you want is called m036infin.hqx. Netscape can easily do this; simply paste the above URL into the "Go To:" field and press return.
2) Use ResEdit to open the ResEdit file Infinity Windoid 2.6.
3) Double click on the WDEF resource that appears in the Infinity Windoid 2.6 window.
4) From the Edit menu, choose Copy.
5) Make sure the Visual Architect application is not running, and open it with ResEdit.
6) Find the WDEF resource within ResEdit's Visual Architect window. Open the resource by double clicking its icon.
7) Select the resource numbered 200 and choose Clear from the Edit menu.
8) Now choose Paste from the Edit menu. A resource with the name "Infinity Windoid 2.6" should appear.
9) Make sure this new resource is selected and choose Get Resource Info from the Resource menu.
10) Change the ID field to 200 and un-check "Purgeable."
11) Close the ResEdit windows and save the changes when prompted. Visual Architect is now ready for use.

Special thanks to Mark Baldwin, Ted Flug, Richard Hill, Matthew Hopkins, Steve Howard, Scott Morison, Kevin Quah, and Steve Wolf for their contributions to this article.

Want to share a tip with the community and get paid for it? Send it in to <mailto:tips@mactech.com>

By Nicholas C. "nick.c" DeMello

Shall we play a game?

Starting Points

If you're interested in touring the online world of Macintosh game programming, to get started I recommend pulling in to Sean M. McDowell's Bitstop. Sean's Mac Game Programmers' Bitstop provides links to the latest news and some of the most essential sites for Mac game programming.

Mac Game Programmer Bitstop
<http://www.undergrad.math.uwaterloo.ca/~smmcdowe/Bitstop.html>

SpriteWorld, the Sequel

Does your program use animation? If so, you probably know about Tony Myle's SpriteWorld, one of the most prominent Macintosh freeware libraries for generating sprite animation's. However, you might not have heard about SpriteWorld 2.0. This is the latest major release of SpriteWorld brought to us by Karl Bunker <mailto:KarlBunker@aol.com> and Vern Jensen <mailto:Vern_Jensen@lamg.com>. SpriteWorld 2.0 introduces new routines that allow you to scroll a large background across your window without having to create a huge GWorld. Imagine what your about box could look like if you made use of Nintendo quality scrolling backgrounds...

Version 2.0 also introduces new "tiling" routines, which allow you to layer sprites so that a closer one passes in front of another sprite. With this release, SpriteWorld even has it's own web site (I'm proud to tell you I was visitor #00000009.) Check out the SpriteWorld pages for documentation, to download the libraries and source code, or just to check out some very impressive demos.

Sprite World 2.0 Pages
<http://users.aol.com/spritewld2/>

Finding Answers

The best resource a starting programmer can have is a good book. Two of the most popular reference books for Macintosh game programmers are Black Art of Macintosh Game Programming and Tricks of the Mac Game Programming Gurus. To help folks get an idea of what each of these books has to offer the publishers have put brief descriptions of the books online.

Matthew Xavier Mora is just beginning to build the "comp.sys.mac.games. programming.book" pages — an HTML format book that will act as a tutorial for Macintosh game programming. A more complete "HIT" format Mac game programming FAQ maintained by Joshua Grass is called the Macintosh Game Programming Workshop.

Comp.sys.mac.games.programmer is the usenet forum dedicated to Macintosh game programming, and the place to look for the comp.sys.mac.programmer.games FAQ — maintained by Kyle Ellrott. Rec.games.programmer and comp.graphics.algorithms are also useful resources, and folks can subscribe to Apple's Macintosh Games Developer mailing list by sending email to <listproc@solutions.apple.com> with "SUBSCRIBE mac-games-dev <your name>" in the body.

Black Art of Macintosh Game Programming Page
<http://www.mcp.com/waite/waite/books.new/Black_Art_Mac_Game_Program/html/bamgpcov.html>

Tricks of the Mac Game Programming Gurus Page
<http://www.mcp.com/hayden/mac_game-gurus/>

Comp.sys.mac.games.programming.book
<http://www.best.com/~mxmora/c.s.m.g.p.b.0.html>

Macintosh Game Programming Workshop
<http://anytime.cs.umass.edu/~jgrass/MGPW/index.html>

The Best of All Worlds

Some will tell you that the Mac can be all things to all people — including a Nintendo system for those of us addicted to Mortal Kombat. If you doubt them, check out John Stiles Macintosh emulators page. John lists twenty five platforms that can be emulated with a Macintosh — everything from an Atari 2600 to a ZX Spectrum. Each of these links takes you to a summary page describing what emulation programs are available, listing important information, and providing links to related sites. For example, the Apple II page begins with a link to an Apple II emulation FAQ; it compares the features and stability of two Apple IIe emulators; provides links for downloading 68k and PPC versions of Apple II+, Apple IIe, and Apple IIgs emulators; and wraps up with links to Apple II software archives.

It is comforting to know that our favorite platform can generate a working environment for testing Magic Cap utilities, a playground for enjoying our favorite Game Boy programs, or even a shell for using TI-81 or HP48 calculator packages. However, it can also provide us with a link to the past, allowing us to remember our first experience with a TRS-80, Commodore 64, or an ancient environment called "Windows."

Macintosh Emulators Page
<http://www.cs.csubak.edu/~jstiles/emulator/> MT

NEWSBITS

By Jessica Courtney



Conix Graphics announces OpenGL for the Macintosh

Conix Graphics announced the release of OpenGL version 1 for the Macintosh. OpenGL assists graphics developers in CAD, Mechanical implementation, Architectural Design, Medical and Scientific Imaging, Geophysical Sciences, Animation, Photorealism, Data Visualization, and Simulation.

OpenGL is the premier 2D and 3D graphics library that allows developers to build geometric models, view models interactively in 3D space, control color and lighting, manipulate pixels, and perform such tasks as alpha blending, anti-aliasing, depth cueing and texture mapping.

Early benchmark testing shows that OpenGL for the Macintosh out performs OpenGL offered by Microsoft and IBM.

Cross platform capability enables programmers the flexibility to transition code easily and smoothly from one platform to another.

OpenGL for the Macintosh by Conix also supports C, ADA, and FORTRAN. Integration with Tenon's MachTen UNIX and Hardware acceleration through RAVE (Rendering Abstract Virtual Engine) will be available first quarter of 1997.
<http://www.conix3d.com>

PKZIP Support to Power Mac

Developers are able to incorporate PKZIP implode and explode capabilities into their programming. The PKZIP DCL provides a library of routines for the compression and decompression of data on a Macintosh or Power Macintosh. It allows programmers to compress and decompress data using their own file handling routines to read and write their data.

Other features include: an all purpose, patented data compression algorithm that compresses both ASCII and binary data quickly, application controlled input/output and memory allocation for the utmost flexibility, and adjustable dictionary size that allows the software to be fine tuned for maximum speed or compression.

ASi is the global leader in compression solutions. Their line of compression utilities includes such names as PKZIP, MultiZip, NetSqueeze, and Nico Mak Computing Inc.'s WinZip, and supports AS/400, DOS, Macintosh, MVS, NetWare, OS/2, UNIX, VMS, VSE, and Windows platforms.
<http://www.asizip.com>

MathSoft Ships Mathcad Plus 6 for the Macintosh

MathSoft, Inc. announces shipment of Mathcad Plus 6 for the Macintosh. An advanced version of the Mathcad technical calculation software package that offers a wide range of new features making it even easier to use for Macintosh customers.

In addition to its technical calculation capabilities, Mathcad Plus 6 for the Macintosh now features built-in Internet connectivity that will benefit users in the technical and academic markets

Mathcads distinctive *live* document interface lets users integrate text, calculations and graphs, then document and share results in an easy-to-use environment. Mathcad Plus 6 also allows users to link and hyperlink worksheets over corporate intranets, WWW or local web servers for applications such as workgroup sharing in a corporate or scientific environment and distance learning in education.

While Mathcad Plus 6 for the Macintosh includes all the features of Mathcad Plus for Windows, and is compatible with the Windows version, the new product also includes several new features including: improved editing and formatting capabilities; simpler menu structures; the ability to create and embed animation within a worksheet; and QuickPlot, a new intelligent formatting capability allowing easy, direct graphing of formulas and expressions. The software, designed for the PowerMac, also provides support for 68030- and 68040-based systems.
<http://www.mathsoft.com>

Claris Corporation Expands Claris Solutions Alliance

Claris Corporation announces the expansion of the Claris Solutions Alliance (CSA), a consortium of software professionals, publishers, authors and trainers who use and recommend Claris software products. Spurred by the fast growth of the CSA, which has risen to nearly 1,500 members worldwide, Claris is adding two new tiers to the organization.

Claris Partners, a new CSA membership, is for highly experienced software development professionals who will work closely with both Claris Corporation product development and sales teams to better address the needs of Claris software customers. In addition to access to beta versions of Claris software, direct technical support, special sales incentives, and numerous co-marketing opportunities, Claris Partner CSA members will undergo reference verification and annual testing by Claris and will receive an official Claris Partner logo for use within all promotional materials.

CSA Enterprise membership, another new category of CSA membership, is designed for a wide variety of professionals within businesses and corporations who are responsible for in-house database development, administration of FileMaker Pro databases, and help desk personnel supporting FileMaker Pro users.

CSA Associate membership, formerly known as the single CSA membership, has been expanded to a include wide scope of users, including software consultants, publishers, trainers and the novice developer/consultant.

Claris Solutions Alliance was founded in 1991. Last June, Claris sponsored the organization's first worldwide developer's conference held in Santa Clara, Calif. The CSA plans to hold its second annual conference this Spring in the San Francisco Bay area. Further details on the 1997 CSA Worldwide Developer's Conference will be provided at later time.
<www.claris.com>

S-CASE Now Supports UML 0.91

MultiQuest announces the release of S-CASE 3.0 (rev. 1.9) with enhanced support for Unified Modeling Language 0.91. This new version reaffirms MultiQuest's commitment to the UML notation proposed by renowned object-oriented methodologists Grady Booch, Ivar Jacobson and Jim Rumbaugh.

S-CASE 3.0 was designed from ground up to provide the true UML metamodel as its foundation. This metamodel is fully accessible using Tcl scripts, thus providing hooks for code generators, report writers and other useful tools. Additional enhancements in this new version include improved code generation and reverse engineering.
<http://www.multiquest.com>

Claris Home Page 2.0 Now Available for MacOS and Windows 95/NT

Claris Corporation announces the availability of Claris Home Page 2.0, the next version of the company's popular web-authoring software for MacOS and Windows 95/NT users.

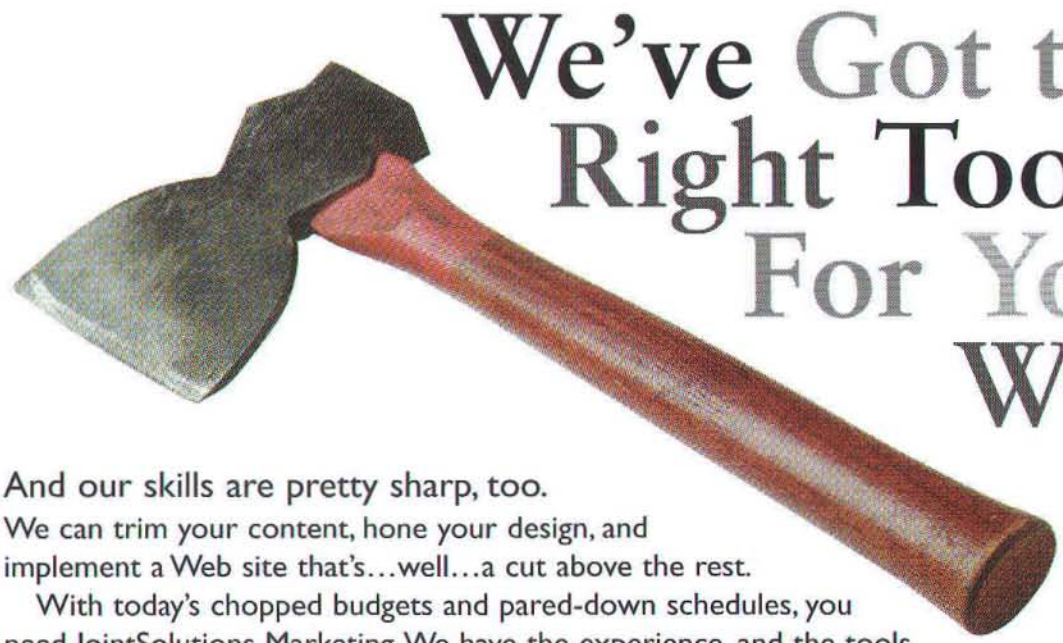
Claris Home Page 2.0, designed for novice-to-expert web page authors, is the essential WYSIWYG, web-authoring solution that provides the ability to design and develop powerful, customized web pages in minutes — without having to learn HTML or programming.

Claris Home Page 2.0 also delivers all the capabilities and tools web masters need — frames, tables, libraries, multimedia plug-in support, built-in remote site publishing, the ability to edit raw HTML, and more. Also, in a cross-platform solution that provides incredible ease of use as well as the flexibility and power that experienced users want.

Claris Home Page 2.0 delivers important new capabilities and features designed to enhance both the look of web pages and the productivity of web authors.

A new feature that novice users will especially appreciate are the 25 site templates provided for the creation of a personal page, calendar, newsletter, presentation, or small business presence. These easy-to-use templates increase productivity, offering multiple style variations for a total of almost 100 individual pages.

Other new features include built-in remote site publishing, with auto-consolidation of files and one-step uploading to a designated server; support for popular multimedia plug-ins, including QuickTime, Java and Shockwave; spell checking and



We've Got the Right Tools For Your Web Site!

And our skills are pretty sharp, too.
We can trim your content, hone your design, and implement a Web site that's...well...a cut above the rest.

With today's chopped budgets and pared-down schedules, you need JointSolutions Marketing. We have the experience, and the tools, to make sure your World Wide Web pages aren't...you know...dull.

We even have a Web server, so site maintenance and updates don't slice into your work time.

Call us today and we'll take a whack at your Web needs.

JOINT SOLUTIONS MARKETING
JointSolutions Marketing
Tel: (408) 471-1500
E-mail: info@jointsolutions.com
http://www.jointsolutions.com

support for multiple fonts within web pages via the FONTFACE tag; client-side, as well as server-side, image maps for easy linking to other pages and sites; and HTML editing enhancements such as syntax coloring.

Claris Home Page 2.0 users will also be able to dynamically re-size and reposition frames by pointing and dragging with their mouse; preview background GIF images from within Claris Home Page; drag and drop tab-delimited text from spreadsheet and database documents to automatically create tables within their web page; and specify row heights and column widths within tables by typing values or dragging cell borders.

Version 2.0 provides concurrency or "selection preservation" between WYSIWYG and HTML modes, allowing users to go to the precise location they were working in the previous mode.

Claris Home Page for MacOS requires a Macintosh, Power Macintosh, or compatible computer with a 68020 processor or higher; 8 MB memory; System 7.1 or later; and 2MB free for minimum installation (6MB for full installation).

For Windows users, Claris Home Page requires an Intel 486 or higher compatible PC System; Windows 95 or Windows NT 3.5.1 or newer; 8MB RAM (16 MB for NT); and 4 MB free hard disk space.
<http://www.claris.com>

Lasso 1.1

Blue World Communications, Inc. announces the release of Lasso 1.1, an advanced tool for integrating Claris Corporation's FileMaker Pro 3.0 for Macintosh databases with the web server.

Lasso is designed for fast performance. Searching FileMaker Pro Macintosh relational databases with tens of thousands of records in under a second is now possible with Lasso.

Lasso operates by recognizing special commands placed in standard HTML files. These commands define all interaction between FileMaker Pro and a MacOS web server. Lasso functions as the "glue" between FileMaker Pro and the web. Lasso HTML files (or format files) are easily edited in HTML authoring programs such as Claris Home Page with drag-and-drop control of data using the supplied FM Link tool. Template files are provided for setting up web databases in minutes.

Lasso allows complete flexibility in how FileMaker Pro data is formatted without requiring any changes to the actual database. Lasso facilitates remote administration of web database configurations from anywhere on the Internet. Lasso also provides routines for security, thus allowing greater control over how databases are served and administered in shared environments.

Lasso offers many features including support for relations/portals, repeating fields, dynamic population of value lists, built-in email forwarding, If... Else... conditional statements, built-in remotely configured security and more.

Version 1.1 adds support for serving images directly from FileMaker Pro, passing tokens (critical for user tracking schemes such as online shopping carts), ScriptMaker scripts, dynamic control of user and browser information, field and record level security, operation as a WebSTAR plug-in and provides enhanced capabilities throughout Lasso's core features.
<http://www.blueworld.com/lasso/> MT



TIPS & TIDBITS



By Steve Sisak

I hope this surprises at least some readers and helps avoid a bug or two. Here's a self-test to see what you know about subtraction. Be honest.

On a PPC chip, which of the following sets the carry bit (assuming an instruction form that sets the carry is used)?

a) 0 - 0
b) 0 - 1

On a 68K chip, which sets the carry?

c) 0 - 0
d) 0 - 1

Did you answer (a) and (d)? That's right. Subraction is defined differently on PPC than on 68K, which might catch you off guard, especially if you're an old hand at 68K asm.

On 68K, sub d0,d1 is defined as d1 = d1 - d0, and the carry behaves like a simple borrow. However, PPC makes direct use of the definition -n = NOT(n) + 1 and, subf r5,r3,r4 is defined as r5 = r4 + NOT(r3) + 1, making the XER[CA] bit (for instructions that update it) reflect an addition carry instead of a borrow.

This difference propagates to extended subtraction. You would do extended subtraction similarly on either chip.

68K:
1) sub low words.
2) subx high words.
PPC:
1) subfc low words.
2) subfe high words.

However, the internal details are different.

subx d0,d1 --> d1 = d1 - d0 - X bit.
subfe r5,r3,r4 --> r5 = r4 + NOT(r3) + XER[CA].

There are many other examples where the carry works completely differently on the two chips (e.g., arithmetic right-shifts).

68K: carry is set (for any type of source) if and only if the LAST bit shifted out of the right was a one.

PPC: carry is set only if the source was negative and ANY ones are shifted out of the right.

Yes, it's all in the manuals, but who reads manuals?

<billKarsh@aol.com>

1) When patching a libraray, _Open needs more param block fields cleared than does _OpenDF.
2) when changing the font of a (popup) menu by hooking the MDEF, LastSPExtra must be set to -1 to clear the font cache both before calling the MDEF and after restoring the font stuff.

Tony Nelson
<tonyn@tiac.net>

Having trouble getting Apple Menu Options' hierarchical menus to show? You may not have called MaxApplZone() early in your application. The various patches AMO installs require sufficient heap to draw the menus; if you don't expand your heap to its maximum early on, the patches may not have the right amount of memory available when first called, and your hierarchical Apple menu will not draw.

Brian Bechtel
<blob@apple.com>

Apple's System 7.5.3 (aka System 7.5 Update 2.0) adds a slick new way to quickly get around while in a StandardFile dialog. While the dialog is up, click on the desktop or any folder open on your desktop. You did not receive the annoying beep you expected as in 7.5.1 and earlier — instead, StandardFile took you right to that directory! This is invaluable when you have your deeply-nested work folder opened on the desktop and StandardFile decides to stick you in some remote location. Now anything visible is one click away! If Apple would give us the new movable modal file dialogs, we could really take advantage of this. These are coming in Copland, but I am hoping they will appear in System 7.6 that has OpenDoc built-in.

Bill Hayden
<mailto:nikol@codewell.com>

MT

Send us your tips or we'll install EvenBetterBusError on your machine! On the other hand, we might just pay you ***$25*** *for each tip we use, or* ***$50*** *for Tip of the Month. You can take your award in goods, subscriptions or US$. Make sure any code compiles, and send tips (and where to mail your winnings) to our* ***Tips e-mail address*** *at tips@mactech.com. (See page 2 for our other addresses.)*



By Edward Ringel

Macintosh Programming FAQs

Lots of information in a good package

Needed: A Guide for the Perplexed

Recalling my undergraduate course in organic chemistry, my textbook went through a rather lengthy description of a synthesis. I would read it and I would think I understood it. I'd then go on to the next page, and an equally daunting process would be left as "an exercise for the reader," implying that at this point the logic should be pretty straightforward and any dummy should be able to figure it out. It would be at this point that I would realize that I wasn't as smart as I thought.

So it is with Mac programming. The first time I tried to put a list in a dialog I carefully read through Inside Macintosh, Volume IV, and I crashed and burned when they started talking about event filter procedures and drawing procedures, with implementation left as an exercise for the programmer.

Things are different from 1986 and 1987, when the List Manager was young, and System 3.2/Finder 5.3 was new and exciting. There are numerous books and good examples of introductory techniques. There are very good guides to the proper construction of a Mac GUI, using standard design elements. Apple has improved its sample code considerably, and I would argue that it is fairly easy to get started if you are willing to spend the time.

The road gets rockier when we get to the next level of sophistication. How do you customize a menu? How do you get the Notification Manager to notify the user? How do you add the little things that make a program special and professional? Certainly, there is an accumulated lore and body of experience which is available on the Internet, in various college and university repositories, the Apprentice CD series, and even from Apple. However, the information is diffuse, often poorly documented, and highly variable in quality and content. For the "every man" mainstream programmer, the person using an unmodified MacHeader straight from Metrowerks or Symantec, bootstrapping to the next level of sophistication can involve an unreasonably steep learning curve.

Enter Mac Programming FAQs

Mac Programming FAQs, by Stephen Baker with Dave Mark, is this needed organized repository of more advanced programming techniques. It is a wonderful book. It consists of a 600 plus page book plus a CD (we'll get back to the CD contents in a bit.) After an introduction, the book consists of 650 frequently asked questions about Macintosh programming, each answered concisely and usually illustrated with code.

I enjoyed this book for several reasons. First, it is simply well written. Baker and Mark are a good team, and they used a terse, but easy to read style. There was little ambiguity in the presentation, but at the same time it was not technical or handbook-like; the information flowed freely from the page. Second, it is written in a single voice; the approach is that of one individual, and it is easy to stay tuned to the presentation. Third, the information was accurate. While there were a few typos and mis-statements, there were very few errors of fact. Fourth, this book is a gold mine of information. I learned a

Ed Ringel is the Contributing Editor for product reviews for *MacTech* Magazine. In his spare time, he is a respiratory and critical care doctor in Waterville, Maine. He can be reached at eringel@mint.net.



2X1

Here's a limited time offer that'll have you seeing double. Order the MacTech CD-ROM™ Volume 11 and get the upgrade (expected release 1Q97) absolutely free! We'll even throw in the shipping costs for the upgrade. And as if that wasn't enough, Volume 11 is discounted too – **now just $79!**

That's right, so for less than you would normally pay for one CD... you'll get two! So hurry, pick up the phone, fire up the e-mail, launch that fax machine, or simply drop by our Web site and score yourself this double-whammy of a deal!

MacTech CD-ROM Vols. 1-11

- 1420+ articles from all 127 issues of MacTech Magazine (1984-1995)
- Improved hypertext and new THINK™ Reference Viewer 2.2
- 100+ MBs of source code
- Full version of THINK Reference 2.0
- 80MBs of FrameWorks/SFA archives
- Sprocket™! MacTech's Tiny Framework that compiles and supports System 7.5 features
- The best threads from Macintosh programmer newsgroups, plus thousands of notes, tips, snippets, gotchas and much more!

Developer DEPOT

Web Site: http//www.devdepot.com • E-mail: orders@devdepot.com

Phone: 1-800-MACDEV-1 • Outside the U.S. & Canada: 805-494-9797 • Fax: 805-494-9798

heck of a lot of new stuff, and got a bunch of questions on some murky topics (for me, at least) answered concisely.

The book purports to address the needs of a wide spectrum of programming skill. While a few of the questions are those posed at the beginner level, the bulk of the book addresses advanced and intermediate programming techniques. I suspect that a beginning programmer who tried to attack this book would certainly get something out of it, but the experience would be akin to learning to swim at the deep end of the pool.

Topics

This book covers questions about mainstream programming. It *does not* cover networking issues, writing PCI card drivers, maximizing efficiency of QuickDraw GX printer drivers, etc. It *does* cover events, dialogs, lists, menus, printing, notification, windows, resources and the like. The book is organized along the lines of the various Managers. Typical questions include:

"How do I get the full path of a file referenced by a vRefNum, parent directory ID, and filename?" (File Manager.)

"How do I attach an icon to an application?" (Finder and Desktop.)

"How do I change the title of a menu in the menu bar after it has been installed?" (Menus.)

"Why can't I access QuickDraw globals like 'thePort' anymore?" (QuickDraw: Drawing.)

"Given a handle to a resource, how can I determine which resource file it came from?" (Resources.)

"How do I draw dimmed (grayed out) text?" (Text: Fonts and Drawing.)

"How do I put a list in a Macintosh model dialog box?" (Lists and the List Manager.)

Some only take a line or two of text to answer. Others, such as the list question, took three or four pages. In general, I had a pretty good idea of how to do something once I had read the answer to the question. Questions are cross-referenced, with respect both to specific FAQs and other more general topics. The only serious criticism I have of this book is that it tries to address the Apple Event issue. I have found this to be one of the most difficult areas of mainstream Macintosh programming, and it seems almost silly to have a few pages of FAQs about this incredibly complex Manager. Fortunately, the authors have the sense to direct the reader to more appropriate, in-depth resources, and acknowledge the limited scope of coverage in the book.

Code

The great majority of articles include some code, even if it just shows how to call a Toolbox function. The code varies from a snippet to a short program, but in my experience it a) always illustrated the point and b) worked when ran. Examples tended to be simple enough to follow, but complex enough not to be trivial and to illustrate the demonstration points successfully. Baker and Mark resisted the temptation to throw in tricks and to hot dog; I think Apple would approve of just about every technique demonstrated. All coding and examples are in ANSI C. There is a short series of FAQs about Pascal versus C, which was the only departure from this observation.

A CD Too!

The CD has three components. First, it contains complete projects (Code Warrior) for the longer examples. This means source, project file, and resource, as well as a compiled executable. Second, snippets are contained within Simple Text files for use in any environment. Third, there is a searchable database that includes every question asked in the book, with its answer, similarly cross referenced. As I said above, I experimented extensively with the code, compiled and ran as advertised; the quality assurance on this material was very good. The database worked well and delivered as promised. If you like working from the screen and using a search engine better than using a book and an index, you'll be very happy with the database.

Can I Ask for Better?

Not really. Frankly, I think this is the best forty bucks I've spent in a long time. I feel as though I have the resources to make my programs more professional in appearance, and I will waste less time in future projects. Indeed, I suspect that it is people such as myself, the serious hobbyist/intermediate level programmer who will find this book the most valuable, precisely for the reasons I just stated. Enjoy!

Mac Programming FAQs, by Stephen H. Baker with Dave Mark. IDG Books, Foster City, CA. Copyright 1996. ISBN 0-7645-4001-7. List price $39.95. MT

Want to know what products are available for MacOS development? Check out Developer Depot™ <http://www.devdepot.com>



(...and we really mean now.)

Think of it as your source of news and announcements in the MacOS developer community. Loaded with up to the latest news, and constantly updated with developments in our industry.

MacTech NOW™ brings you up to speed on everything you need to know – instantly! And thanks to our new "fast-download" design, you'll get to the information you want in seconds.

Give it a spin! Check it out today and get access to over 1500 pages loaded with news, tips, programming secrets, product reviews, and much more. And for those of you looking for some kicks, there's the ever popular "Programmer's Challenge" section where you'll get to bang heads with the best in the community.

Log on to MacTech NOW. Things are happening right now, and you should be aware of them.

http://www.mactech.com

MacTech, MacTech Magazine, MacTech CD-ROM, MacTech Web, THINK Reference, Developer Depot, Sprocket, JavaTech, WebTech, BeTech, Developer Central, Virtual Developer Central and the MacTutorMan are trademarks of Xplain Corporation. Other trademarks and copyrights appearing in this printing or software remain the property of their respective holders.

ADVERTISER/ PRODUCT LIST

LIST OF ADVERTISERS

Adianta Inc. 40
The AG Group 53
Aladdin Knowledge Systems Ltd. 5
Aladdin Systems, Inc. 20-21
Bare Bones Software, Inc. ..... IBC
BeeHive Technologies, Inc. 11
Bowers Development 23
Conix Graphics 40
Datapak Software Inc. 41
Developer Depot 72, 78
Faircom 8
Fortner Research, LLC 32
Jasik Designs 39
JointSolutions Marketing 71
MacTech CD ROM 75
MacTech ™ Now! 77
Mathemaesthetics, Inc. ..... IFC
Metrowerks ..... BC
Micro Macro Technologies, Ltd. 17
Neologic Systems 25
Onyx Technology 7
PACE Anti-Piracy 67
Parallel Staffing Group, Inc. 49
Partium, Inc. 29
Perforce Software 33
Quasar Knowledge Systems 27
Ray Sauers Associates 66
Scientific Placement 49
Seapine Software, Inc. 27
SirS 49
SmartCode Technologie 15
Stone Tablet Publishing 57
Water's Edge Software 38
Willows Software, Inc. 1

LIST OF PRODUCTS

DB I/O • Beehive Technologies, Inc. 11
AppMaker • Bowers Development 23
BBEdit • Bare Bones Software, Inc. ..... IBC
CodeWarrior™ • Metrowerks ..... BC
The Debugger V2 • Jasik Designs 39
Developer Tools • Developer Depot 72, 78
DragInstall • Ray Sauers Associates 66
EtherPeck™ • The AG Group, Inc. 53
FairCom Developer CD • Faircom 8
InstallerMaker • Aladdin Systems, Inc. 20-21
MacHASP • Aladdin Knowledge Systems Ltd. 5
MacNosy • Jasik Designs 39
MacRegistry™ • Scientific Placement 49
The Memory Mine™ • Adianta Inc. 40
MicroGuard Plus™ • Micro Macro Technologies, Ltd. 17
neoAccess™ • Neologic Systems 25
ObjectSet Mail SDK •SmartCode Technologie 15
Open GL® for the Macintosh • Conix Graphics 40
PAIGE • DataPak Software, Inc. 41
Perforce® • Perforce Software 33
PowerTap • Fortner Research, LLC 32
Recruitment • Parallel Staffing Group, Inc. 49
Recruitment • Scientific Placement 49
Recruitment • SirS, Inc. 49
Resorcerer ® 1.2 • Mathemaesthetics, Inc. ..... IFC
SmallTalk Agents® • Quasar Knowledge Systems 27
Spotlight ™• Onyx Technology 7
Stone Table • Stone Tablet Publishing. 57
TestTrack™ • Seapine Software, Inc. 27
Tools Plus™ 3.0 • Water's Edge Software 38
Twin APIN • Willows Software Inc. 1
WinRez • Partium, Inc. 29
Worldwide Web Pages• Joint Solutions 71

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

Developer DEPOT™

- Web site: http://www.devdepot.com • E-mail: orders@devdepot.com
- Phone: 1-800-MACDEV-1 (Outside the U.S. & Canada: 805-494-9797)
- Fax: 805-494-9798

✓ **Hundreds of developer products**

✓ **Satisfaction and lowest price guaranteed**

✓ **World renowned customer service**

✓ **Order by phone, E-mail, fax or through our continually updated Web site**

NOW CARRYING APPLE PRODUCTS!

see pages 3-4



Open Programmer's Hours...



...and all day too. In fact our Web site is open 24 hours a day, 365 days a year!

Which means that whenever you get a craving for the latest in developer tools, we're just a couple of keystrokes away.

You can browse for hours (we don't mind), pick up the items you want, and conveniently pay with your major credit card at our virtual check-out. Then, kick back and wait for your goodies to arrive – all without ever leaving your house!

Best of all, you'll shop confidently, because Developer Depot guarantees your complete satisfaction and lowest price for 30 days after your purchase.

Talk about a safety net!

That's right, and speaking of nets, get connected and come check out the vastest on-line selection of developer products... any day, any time!

Developer DEPOT™

http://www.devdepot.com

Orders by phone can be placed M - F from 9:00 a.m. to 5:30 p.m. pacific time at 1-800-MACDEV-1.

ANNOUNCEMENT

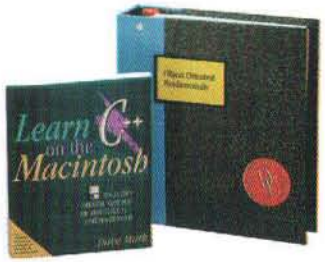
Developer DEPOT

now carries Apple brand products

See pages 3-4









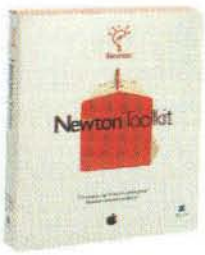


TABLE OF CONTENTS

02 MACTECH

03 APPLE PRODUCT

05 DEVELOPMENT ENVIRONMENTS

08 INTERNET RELATED

09 SCRIPTING

10 TOOLS, LIBS & UTILITIES

16 BOOKS & REFERENCE

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

One & 2-Day SHIPPING (call for details)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Developer Depot 30 day Money Back, Price and Satisfaction Guarantee

Developer Depot products are sold with a 30 Day money back guarantee on user satisfaction, lower prices and against defects. If, for any reason, you are not satisfied or find the same product at a lower price within 30 days, please call Customer Service at 800-MACDEV-1 and request a Return Merchandise Authorization (RMA) number to get a full refund or the difference in price (where applicable). You must return undamaged product at your expense, including all its original packaging, documentation and the blank warranty card if applicable. Developer Depot will replace defective product upon receipt of the defective merchandise. Please remember to back up your data before installation of any new hardware, software, or peripherals; we cannot be responsible for any lost data. Policies, item availability, and prices are subject to change without notice. The price in effect when we receive your order will be the price that you are charged. We are not responsible for any typographical errors in this or any other catalog, nor for any misstatements from any vendor. Purchase orders are also accepted but must be in writing, signed, come with a contact person and a telephone number, and mailed to P.O. Box 5200, Westlake Village, CA 91359-5200. Faxed copies and purchase order numbers alone are not acceptable.

Stuff our lawyer made us write.
Developer Depot makes no other warranties. All other warranties, either expressed or implied, including the implied warranty of merchantability and fitness for a particular purpose are disclaimed. Developer Depot shall not be liable for any direct, special, incidental or consequential damages including lost profits, from any delay in delivery, or for any personal injury arising from the use of any product sold through Developer Depot. The limit of direct damages, if any, shall not exceed the purchase price of the product.© 1997 Xplain Corporation. All rights reserved. Any unauthorized duplication is in violation of federal laws. Developer Depot is a registered trademark of Xplain Corporation. All product names in this catalog are the trademarks of their respective holders.

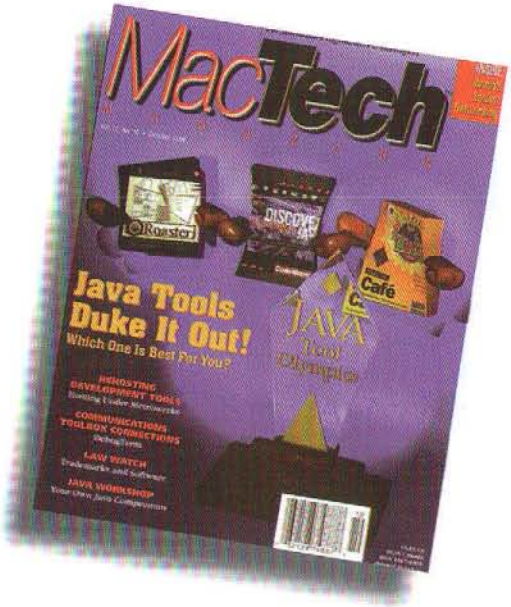
© 1997 Xplain Corporation. All rights reserved. Any unauthorized duplication is in violation of federal laws. Developer Depot is a trademark of Xplain Corporation. All product names in this catalogue are trademarks or registered trademarks of their respective holders

Check our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com

For Macintosh Programmers & Developers

MacTech™ MAGAZINE

Subscriptions: US magazine: **$47** for 12 issues (MTYRDM)
Canadian: **$59** for 12 issues (MTYRCM)
International: **$97** for 12 issues (MTYRFM)
Back Issues: **$10** each (subject to availability)





Limited time offer: Buy **Volume 11** ***NOW***, and get a **free** upgrade to the soon-to-be-released **Volume 12!** (Free shipping on the upgrade!)

MacTech CD-ROM Volumes 1-11

• 1420+ articles, from all 127 issues of MacTech Magazine (1984-1995). • Improved hypertext, and a new THINK Reference Viewer — for lightning quick access! • New hyperlinks between articles. • 100+ MB of source code — use them in your own applications, with no royalties! • Full version of THINK Reference™ – the original online guide to Inside Macintosh, Vols. I-VI. • 80MB of FrameWorks/SFA archives. The most complete set of FrameWorks archives known. • Sprocket™! MacTech's Tiny Framework that compiles quickly and supports System 7.5 features. • The best threads from the Macintosh programmer newsgroups plus thousands of notes, tips, snippets, and gotchas. • Popular tools that Macintosh programmers use to increase their productivity and much more!

List $89.00 Our Price **$79** (SMTCD11)

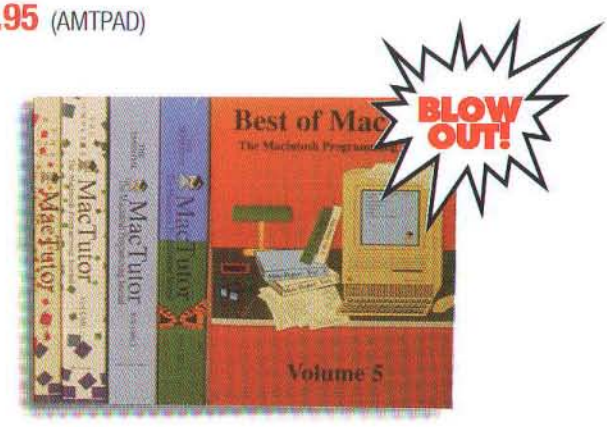
Upgrades: Our Price **$39** (SMTCD11U)



MacTech Mouse Pad

Slide on this! With an extra-large surface (11" by 10") and a deluxe sleek plastic coating, you'll be zooming across your screen in no time at all. Speed limit not enforced!

Our Price **$8.95** (AMTPAD)



Best of MacTutor

The Best of MacTutor Collection, Vols.3-5.

List $99 Our Price **$9.95** per set (BMTBEST).



AppleScript Software Development Toolkit 1.1

by Apple Computer, Inc.

- AppleScript language, system software extension, and script editor.
- FaceSpan 1.0.
- Developer's redistribution license for AppleScript System software extension and FaceSpan runtime code.

Our Price **$49** (SASDT)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Apple Media Tool Programming Environment 2.0

by Apple Computer, Inc.

- This object-oriented language and application framework allows programmers to customize features used within the Apple Media Tool authoring environment.
- Includes an expanded Apple Media Language (AML) class library, incremental compiling and linking of AML code, faster debugging facilities, Macintosh Programmers' Workshop (MPW), and user-oriented documentation written from an AMTPE developer's perspective.
- Portable across 68K, Power Macintosh, and Windows platforms.

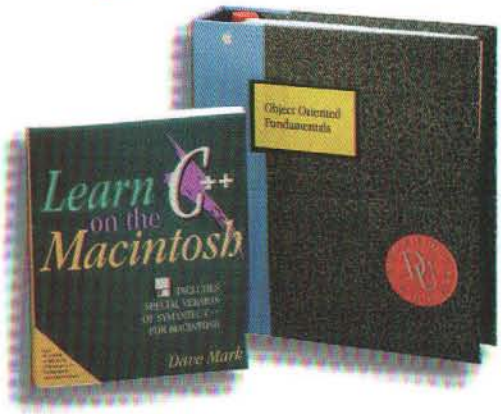
Our Price **$995** (SAMTPE)

Object-Oriented Fundamentals 1.1

by Apple Computer, Inc.

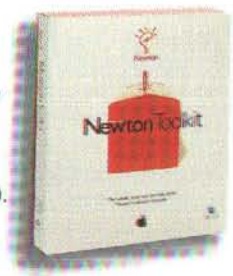
- DU's multimedia Object-Oriented Fundamentals course enables you to easily make the paradigm shift from procedural to object-oriented design. It will introduce you to the entire object skill set, from general concepts through analysis and design. This course includes a "lite" version of Metrowerks CodeWarrior integrated development environment for use in the labs. This course also includes a copy of the book Learn C++ on the Macintosh by Dave Mark.

Our Price **$245** (SOBORFU)



Newton Toolkit 1.6

by Apple Computer, Inc.



With Newton Toolkit, you can easily create software that runs on any Newton PDA, including Apple's MessagePad and Motorola's Marco.

- Now supports Newton 2.0.
- Dynamic Language Eases Development.
- Allows you develop applications interactively.
- New Compiler Enables Faster Applications.

Newton Toolkit 1.6 Our Price **$299** (SNETO)

Newton Toolkit Update 1.6 Our Price **$49** (SNETOUP)

Newton Programmer's Guide for Newton 2.0

by Apple Computer, Inc.

- The Newton Programmer's Guide consists of two volumes covering the Newton System Software, and one volume covering Newton Communications.
- The two-volume set, Newton Programmer's Guide: System Software, is the definitive guide and reference for Newton programming. This set of volumes explains how to write Newton programs and describes the system software routines that you can use to do so.
- The Newton Programmer's Guide: Communications, describes the Newton communications system software for version 2.0.

Our Price **$149** (BNPGFN)

Web site: http://www.devdepot.com • E-mail: orders@devdepot.com

QuickTime VR 2.0 Authoring Tools Suite

by Apple Computer, Inc.

- QuickTime VR is a cross-platform software from Apple which enables webpage designers and professional developers to create new multimedia products and webpages incorporating QuickTime VR content. With QuickTime VR, users interactively navigate through 360° views of space, and explore three dimensional objects on Macintosh or Windows-based personal computers.
- The QuickTime VR Authoring Tools Suite is a set of Macintosh tools to create and link panoramas and objects from photographic, digital, video, or computer generated images.
- Included is a complete set of documentation for planning, designing, photographing, and creating QuickTime VR panoramas and objects. The authoring tools also allow you to link objects to panoramas using clickable hot spots.

Included on the CDs are:

- A software tool (MPW-based) that stitches and blends adjacent images into a panoramic PICT file.
- A software tool (MPW-based) that dices and compresses panoramic PICT files to less than 100 KB (low resolution) per panorama.
- A scene editor (HyperCard-based) to create QuickTime VR scenes by adding and positioning nodes, hot spots, linking nodes together, and for linking QuickTime VR objects to scenes.
- A variety of utility tools for formatting the data into the runtime software.

Due to a revolutionary distortion-correcting algorithm, QuickTime VR panoramas and objects maintain a normal perspective when the user moves the mouse. The speed of the algorithm allows up to 24-bit color images. Both vertical and horizontal panning can occur at fast speeds.

Our Price **$395** (SQTVRATS)

QuickTime Developer's Kit 2.0

by Apple Computer, Inc.

- QuickTime 2.0 Extension, QuickTime Power Macintosh Extension, and QuickTime Musical Instruments extension.
- Utilities like MoviePlayer 2.0, 16-bit Audio Compression, etc.
- Sample content such as MPEG Movies, Music Movies, Time-Code Movies, and 60 field per second movies.
- Includes software-only playback features such as faster 2x playback mode for current compressors, Apple Cinepak compressor, 1-bit fast dithering, network tuning, load-into-RAM option, and Photo CD support.

Our Price **$99** (SQTDK)



Virtual Reality Programming with QuickTime VR 2.0

by Apple Computer, Inc.

- Virtual Reality Programming Book/CD-ROM for QuickTime VR 2.0
- Enables you to write C and C++ programs using QuickTime VR 2.0
- Allows QuickTime VR to be used in games, multimedia titles and other programs.
- QuickTime VR 2.0 objects can be zoomed in on, panned, or linked with hots spots.
- Both panoramas and objects have hot spots linked to World Wide Web URLs.

Our Price **$39.95** (SVRPQT)

Multimedia Authoring with Apple Media Tool

by Apple Computer, Inc.

Apple Media Tool offers new multimedia users a way to get started creating interactive multimedia with minimal learning time. This self-paced tutorial will make Apple Media Tool (AMT) even easier to understand and to use. Using this tutorial, you will create a realistic multimedia project using exciting techniques such as QuickTime movies, animation and more. A demo version of AMT is included and can be used for the exercises. Training Format: Tutorial with labs.

Our Price **$49.95** (SMWAMT)

Apple Dylan

a dynamic object oriented language and development environment

Apple Dylan Technology Release

by Apple Computer, Inc.

- Contains a PowerPC-native prototype version of a development environment based on the Object Oriented Dynamic Language (OODL) Dylan. Developers will be able to produce code targeting both 680x0 and Power Macintosh systems.
- Automatic memory management.
- Application framework and user-interface builder.
- High-level exception handling.
- Cross-language support for C code and APIs.

CD & Online Documentation Our Price **$39.95** (SADTRO)

CD & Hardcopy Our Price **$59.95** (SADTRH)



CodeWarrior 10 Gold

by Metrowerks

- New improved IDE - graphical browser view, global preferences and an improved external editor!
- Full Java toolkit: project manager, linker, editor, class browser, source-level debugger, applet viewer and more!
- Everything you need for industrial-strength programming. Develop for Macintosh, Power Macintosh, Windows 95, Windows NT, Magic Cap and BeOS.
- Award-winning, easy-to-use IDE supports C/C++, Object Pascal and Java.
- Includes online books, extensive reference material and Apple Guide files for easy navigation through tutorials and examples. Two free updates and technical support included with registration.

Our Price **$399** (SCWGOLD)

SEE RELATED PRODUCTS: • Code Manager • Inside Power Plant • Inside CodeWarrior 9 • C++ Programming with CodeWarrior • PowerMac Programming Starter Kit • Discover Programming for Macintosh • Learn C on the Macintosh • Metrowerks CodeWarrior Programming

Discover Programming for Macintosh

by Metrowerks

- Includes full working version of CodeWarrior along with three online tutorial books and Dave Mark's "Learn C on the Macintosh" converted to AppleGuides.
- Includes C, C++ and Object Pascal compilers for generating 68K Macintosh code, source-level debuggers, object-oriented frameworks (PowerPlant, MacApp), Apple's MPW, complete online documentation and source code examples for all languages and platforms.
- The IDE software has been localized in eight languages plus English. This product is not sold as a subscription.
- **Includes a 3 month subscription to MacTech Magazine.**

Our Price **$79** (SCWDISCMAC)

SEE RELATED CATEGORY: Dev. Environments



CodeWarrior Wear (XL only)

You live it, you breath it... you might as well wear it!

- Black CodeWarrior Sweatshirt:
Our Price **$29.95** (ACWSWEAT)
- "Blood, Sweat & Code" black short-sleeve shirt:
Our Price **$9.95** (ACWSBLOOD)
- Hawaii Five-0 shirt:
Our Price **$7.95** (ACWHAWAII)
- Arnold at work T-shirt:
Our Price **$9.95** (ACWARNLD)
- Hat: Our Price **$14.95**

Please Specify: Black (ACWBHAT) or White (ACWWBHAT)

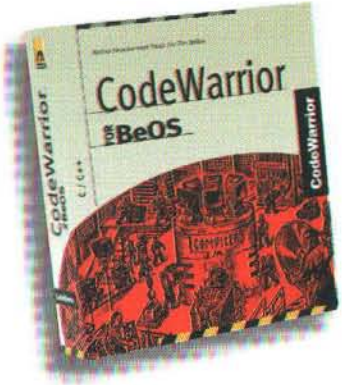
- Winter Hat (see Web site)
Our Price **$14.95** (AWINHAT)

CodeWarrior for BeOS DR2

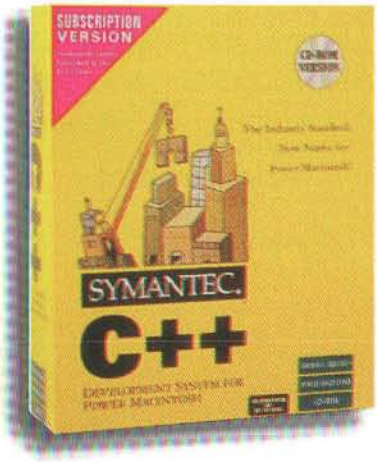
by Metrowerks

- Start programming for the new, innovative Be Operating System (BeOS) with complete set of Codewarrior tools
- BeOS-native Integrated Development Environment (IDE) with all the familiar CodeWarrior features at your fingertips
- A BeOS PowerPC compiler and linker, an editor w/syntax color and styling, and a source-level debugger
- BeOS header and libraries, complete documentation, useful C++ classes, and sample code
- the Be Operating System DR8.2 for Power Macintosh will allow you to run and program for the BeOS on a 603 or 604 PCI based PowerMac

Our Price **$149** (SCWFB)



Web site: http://www.devdepot.com • E-mail: orders@devdepot.com



Symantec C++ 8.5
by Symantec Corporation

- Native for Power Macintosh.
- Develop full-featured Power Macintosh applications quickly and easily!
- Environment includes: a true native Power Mac implementation of the C++ language, MrC/MrC++ compilers for fast Power Mac executable code, Visual Architect for fast, easy GUI generation, a new THINK project management system that supports complex applications, a new editor/browser , a powerful, easy-to-use source code debugger.
- Also includes: The industry-standard THINK Class Library!
- Next two updates free when you send in your registration card and more!

List $499 Our Price **$349** (SSYMCPP)

SEE RELATED PRODUCT: Symantec C++ Programming, Learn C++ on the Macintosh, Programming in Symantec C++, Mastering theTHINK Class Library.

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com

Symantec C++ for 68k
by Symantec Corporation

• The standard in development languages. • Powerful combination of fully integrated visual tools and the latest in C and C++ compiler technology. • Provides an object-oriented approach to application development. • Write code that is extensible, reliable, and maintainable. Includes: Integrated Environment with full source-code debugging, Incremental Linker which eliminates long link times, THINK Class Library, AppleEvents, Visual Architect™, THINK Inspector, Project Models, Source-Code Control with integration with Apple's SourceServer (included), Support for Scripting, Powerful Standard Libraries which includes I/O Streams, ANSI standard C library, and sample programs. • Full source code is included – Create applications that run on 68K Macs and Power Macs (emulated). These applications can easily be migrated to native Power Mac, by trading up to Symantec C++ for Power Mac.

List $299 Our Price **$99** (SSYMCPP68K)

Think Pascal Version 4.0
by Symantec Corporation

- Great for professionals and students
- Fully integrated for rapid turnaround time – take advantage of System 7 capabilities
- Supports large projects, enhanced THINK Class Library, System 7 compatibility, superior code generation, and smart linking.
- Includes four Macintosh disks, a user manual, and an object-oriented programming manual.

Our Price **$165** (SPASCAL)

Power MachTen–UNIX
by Tenon Intersystems

- Dynamically linked shared libraries, memory mapped file access and integrated UNIX and Macintosh development tools.
- BSD 4.4 and conforms to the Federal Information Processing Standard 151-2 (the POSIX FIPS).
- Pre-emptive multitasking for UNIX applications and includes a full featured high-performance TCP/IP protocol stack that supports multi-homing and multi-casting, features not yet available even with Apple's new Open Transport.
- A complete UNIX software development environment with a source-level debugger and C, C++, and Fortran compilers all generating native PPC code.
- Also included is a high-performance X server and complete X11R5 X.

Our Price **$695** (SM10PPC)

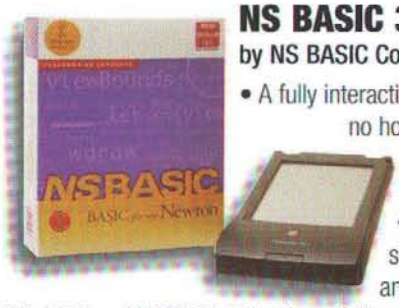
SEE RELATED CATEGORY: Internet Related

Fortran 77SDK
by Absoft Corporation

For Power Macintosh includes a globally optimizing native compiler and linker, native Fx™ multi-language debugger, and Apple's MPW development environment.

• The compiler is a full ANSI/ISO Fortran 77 implementation, and includes all MIL-STD 1753 extensions, Cray/Sun-style POINTER, and several Fortran 90 enhancements • MRWE, Absoft's framework library is included in the MIG graphics library • Supports the native Macintosh PPC toolbox • Includes Absoft's Fx debugger which can debug intermixed FORTRAN 77, C, C++, and PPC assembler • The linker compiler, and debugger all run as native PPC tools, and produce Macintosh PPC executables

Our Price **$699** (SF77)



NS BASIC 3.5 for the Newton with Visual Designer

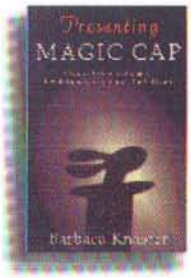
by NS BASIC Corporation

• A fully interactive implementation of the BASIC programming language. • Runs entirely on the Newton – no host is required. • Create files, access the built-in soups, and the serial port for input and output. • Work directly on the Newton, or through a connected Mac/PC and keyboard. • Get the **BASIC Internet Tool**, available at no charge to NS BASIC users from www.nsbasic.com. Release Notes with sample code are available from the same location. • Runs on any Newton MessagePad 130 with NS BASIC and the Newton Internet Enabler. Also runs on MP 1201s with NOS 2.0 that have full memory available. • Write short programs to access News, mail and the web.• Create custom newsreaders that do filtering or archiving • Load a spreadsheet or stock trade triggering program with stock prices from the web. • Send mail containing data collected in an NS BASIC program • Update a database with data sent by email.

List $99 Our Price **$94.99** (SNSBASIC)

Presenting Magic Cap, A Guide to General Magic's Revolutionary Communicator Software

by Barbara Knaster



- Perfect for novices as well as experts who want to take full advantage of Magic Cap.
- Step through its screens, see how it works, and learn ways to use it.
- Describes the power of smart messaging to customize the way you handle e- mail, a name card file that learns how to keep track of your contacts, a datebook that performs like a faithful assistant, and countless other abilities.

List $16.95 Our Price **$15.25** (BPRESMAGIC)

Personal MachTen for 68K Macs

by Tenon Intersystems

Order Toll-free
800-MACDEV-1
(800-622-3381)

- MachTen UNIX for Macintosh (from Classic on up, including PowerBooks and Duos)
- A Mach-based Berkeley UNIX with pre-emptive multi-tasking
- Extends the Mac OS with UNIX networking and software development tools.
- Built-in internet services include domain name service, POP mail service, internet routing, SLIP & PPP, and Web service.

Our Price **$495** (SM10PER) Also Available: MachTen XWindows Our Price **$350** (SMACHX)

SEE RELATED CATEGORY: Internet Related

WAIT... There's More!

Here's a list of all available products.
For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	LIST PRICE	OUR PRICE
Personal MachTen for 68K Macs	SM10PER	$600.00	$495.00
CMaster	SCMASTER	129.95	129.95
CodeWarrior Discover Java	SCWDISCJAVA	99.00	99.00
LPA MacProlog Developers Edition	SLPAD	1500.00	995.00
LPA MacProlog Programmers Edition	SLPAP	745.00	495.00
LS FORTRAN	SLSFORT	695.00	595.00
Mac FORTRAN II	SFORT2	595.00	549.00
Professional MachTen for 68k Macs	SPROM10	695.00	695.00
SmalltalkAgents	SSTA	695.00	695.00

Roaster Subscription

By Natural Intelligence Inc. Get the most out of Sun's new Java™ Programming Language!

- Write, test and run Java applets on the Macintosh in a full-featured Mac development environment
- Features include: project window that includes a finder-like view of packages, Macintosh native compiler, source code editor with powerful search features and intuitive use interface, runtime engine for quick and easy applet testing.
- Requirements: PowerPC based Macintosh, CD-ROM.
- Price includes the current DR 2 release and the next two non-developer releases.

List $399 Our Price **$299** (SROAST)

SEE RELATED CATEGORY: Dev. Environments

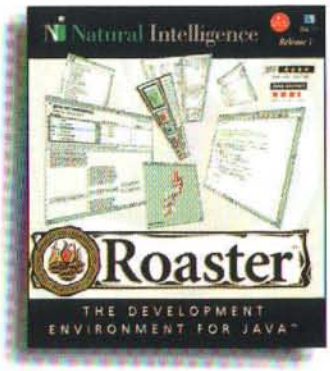
Roaster

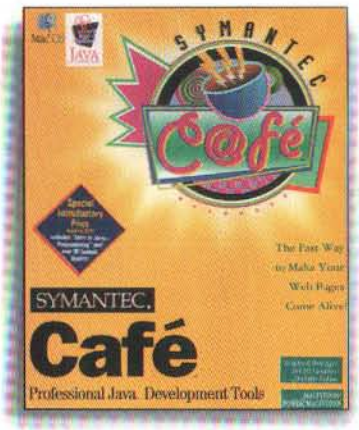
By Natural Intelligence Inc.

- Price includes the current DR 2 release and the first non-developer release.

List $179 Our Price **$129** (SROAST1)

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com





Symantec Café

by Symantec Corporation

Symantec Café for Macintosh is a stand-alone, professional Java development environment that replaces the previously released Symantec Caffeine Java tools. Café is ideal for both novice and professional Java developers. Café includes an "Intro to Java Programming" tutorial, Java language reference, Café Studio visual tools for graphically drawing Java applets on screen and having the corresponding source code generated automatically, and over 90 sample applets that can be easily modified to create your own custom applets.

- For professionals familiar with programming, Café provides an unrivaled feature set, including an integrated graphical class browser, fast native Java compiler, a full-featured project manager, a professional Java source code editor, and full support for AppleScript and AppleEvents for automating time consuming tasks.
- Café comes with one year of free access to Symantec Java Central, including the right to download patches and updates as they become available.

Our Price **$99** (SSYMCAFE). SEE RELATED CATEGORY: Dev. Environments

OOFILE Reporter Writer by A.D. Software

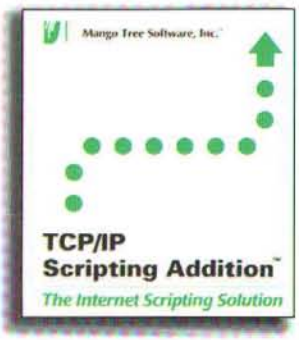
This is a full embedded report-writer, allowing you to preview page-by-page and either print or save as plain text, HTML or RTF. Multiple levels of breaks, database views and headers and footers are provided using a clean object-oriented design. Incudes RAM-based version of OOFILE database. Included in full OOFILE Platform Bundle. Price includes 1-tear subscription. Saving to file without preview of printing is cross-platform-run on your Mac/Win/Unix server and create Web Pages.

Our Price **$500** (SOORW)

TCP/IP Scripting Addition by Mango Tree Software

- Award-winning AppleScript scripting addition
- Allows you to write scripts using MacTCP™ commands in AppleScript™.
- Send e-mail or files through a script, check if users are logged on (via Finger), automate FTP, Gopher, NetNews, Telnet, and LPR, verify links in HTML documents, and quickly write many other TCP/IP client-server programs.
- Works with AppleScript, MacTCP 2.0.4 and Open Transport

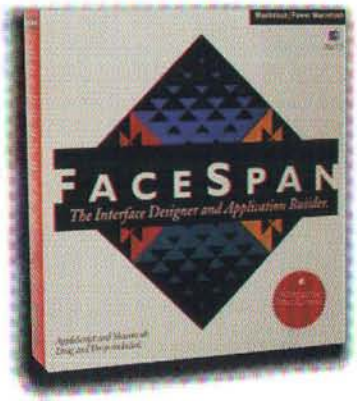
Our Price **$49** (STCP) SEE RELATED CATEGORY: Scripting



WAIT... There's More!

Here's a list of more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	LIST PRICE	OUR PRICE
CodeWarrior Discover Java	SCWDISCJAVA	$99.00	$99.00
SmalltalkAgents	SSTA	695.00	695.00
Tenon Ported Applications CD	SPORTED	50.00	49.95



FaceSpan v2.1

by Digital Technology International

• Develop Integrated Software • Make Stand Alone Applications • Create Friendly Interfaces • Develop Quick Prototypes • Print multiple pages with sophisticated layouts • Script essential elements of the FaceSpan application • Play and record sounds as either "snd" resources or as "AIFF" files. • Create miniature or complete applications that will run on either Power PC or 68k computers. • Use precise time measurement for implementing timed behaviors. • Enhanced Save Options • New Properties • Proportionally scale PICT images • Align images in a pictbox • Monitor and respond to low-memory situations • Automate any application • Increased support for Frontier UserTalk!

List $299 Our Price **$279** (SFACESPAN)

Script Debugger

by Late Night Software Ltd.

- A powerful and flexible AppleScript authoring tool – get the most from AppleScript!
- Advanced debugging environment offers single-step script execution with breakpoints.
- Script Debugger dictionary browser features a graphical view of objects provided by scriptable applications.
- Includes Late Night Software Scripting Additions – a collection of more than 70 new AppleScript commands, and Scheduler, a utility that allows you to launch scripts at pre-determined times.

List $129 Our Price **$119** (SDEBUG)



NEW 2.0!



Scripter 2.0 by Main Event Software

• Whether you're looking for a tool to introduce yourself to AppleScript's unmatched capabilities, or a well-crafted development environment, Scripter, the Script Construction Set, is the obvious choice. FaceSpan 2.x integration: If you use the FaceSpan 2.x interface creation environment from Digital Technology International, you can use Scripter seamlessly instead of FaceSpan's basic editor, to write, test and debug scripts attached to objects in FaceSpan. • Scripter and FaceSpan work together: one click opens your FaceSpan script in Scripter, another sends it back. • Debug handlers without modifying your scripts using the Call Box, a special window for calling individual routines in a script. • Live editing: simulate applet, CGI, FaceSpan messaging Interactively debug live interapplication messages sent to a script application or AppleScript CGI from an applet, your Web server or FaceSpan 2.x. You won't have to modify your Web pages or your CGIs. • ScriptBase is now included with Scripter. Use it to store your data and media elements (frequently used values, text, pictures, scripts, HTML, headers, file references) and share them between scripts all with a special new browser. • Easily write and compile scripts that have handler declarations and other vocabulary specific to a particular scriptable application. • Scripter is the natural companion to AppleScript for users at all levels of proficiency. Don't write scripts without it!

List $199 Our Price **$179** (SSCRIPTER)

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	LIST PRICE	OUR PRICE
DogPatch	SDOGPATCH	$299.00	$199.00
PreFab Player	SPLAYER	95.00	95.00
ScriptBase	SSCPTBASE	59.00	59.00
ScriptWizard	SWIZ	89.00	84.95

Web site: http://www.devdepot.com • E-mail: orders@devdepot.com

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



BBEdit 4.0

by Bare Bones Software

- A powerful, easy-to-learn text editor.
- Adds new features for HTML coders, including a spelling checker and HTML tag palette.
- Accelerated for Power Macintosh; dragging supported everywhere; Internet Config aware; PowerTalk aware.
- Integrated support for Symantec's IDE, Metrowerks CodeWarrior, THINK Reference 2.x, MPW ToolServer, and most other environments.
- Many UNIX style tools, including "grep" searches, file comparisons, and sorting. Multi-file search and replace.
- PopUpFuncs feature lets you jump to a function from a menu.

List $119 Our price **$94** (SBBEDIT)



QUED/M 3.0

by Nisus Software

- The programmer's text editor that defined the industry standard for speed and efficiency.
- PowerPC native.
- Features integrated support for Symantec C/C++, Metrowerks CodeWarrior 6, and MPW.
- Supports all the major development environments on the Macintosh.
- Dozens of powerful editing features, including unlimited undo and redo, macro language, scripting, text folding, ten editable/appendable clipboards, markers, displaying text as ASCII codes, dynamic coloring of C/C++ keywords/comments, rectangular and non-contiguous selection.
- Includes Celestin Company's APPRENTICE 4.

List $149 Our Price **$89** (SQUEDM)

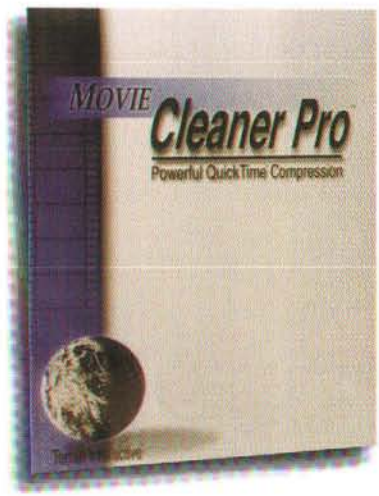
Movie Cleaner Pro 1.2

by Terran Interactive

- Compress QuickTime movies
- Powerful and easy-to-use
- Includes: drag and drop batch processing, suspend and resume, adaptive noise reduction, IMA audio compression, high quality crop and resize, A/V fades, gamma correction, de-interlacing and more!
- Automatically suggests the best settings for your application.
- Great for both beginners and experts
- Essential for CD-ROMs or Web sites

Our Price **$189.95** (SMOVIE)

SEE RELATED PRODUCTS: Quick Time Official Guide

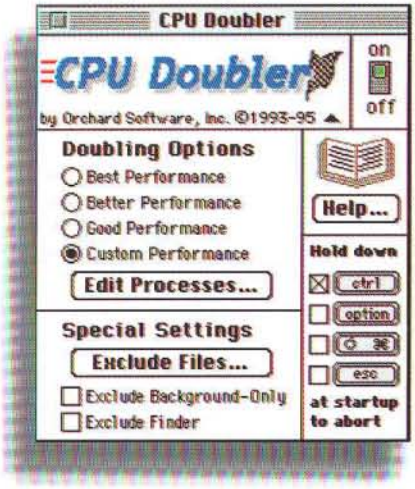




CronManager
by Orchard Software

- Implements the UNIX Cron facility.
- Open any Macintosh file on a given date and time.
- Simple interface.
- Works with any Macintosh file.
- Cron Manager bundled with CLImate,

Our price **$26.95** (SCRONMGR)



CodeManager
by Metrowerks

- Source code control system for the Macintosh.
- Based on and compatible with Microsoft Visual SourceSafe version 4.0.
- Works with any type of binary file including graphic, database, library and executable files.
- Includes support for multi-platform projects, security features, reverse delta versioning of files, comment areas for each revision, project analysis and reporting tools.
- Sold on a subscription basis with two future product updates.
- **Includes a 1 year MacTech subscription.**

Our Price **$399** (SCDMGR)

SEE RELATED CATEGORY: Dev. Environments

CPU Doubler
by Orchard Software

- Performance enhancement utility for the Macintosh.
- Increases the speed of your computer by 100%.
- Works on both the PowerPC and 68K Macintosh.
- Manages computer throughput using a proprietary scheduling algorithm.
- Ensure optimal performance and compatibility.

Our Price **$79.95** (SCPU2X)

Order Toll-free
800-MACDEV-1
(800-622-3381)

NeoAccess
by Neologic

- Full-featured object database engine for use in Macintosh, Windows, Unix and DOS based C++ applications.
- Extended binary trees and binary search algorithms tuned for short access times; dynamically combined, collapsed, and compressed indices; object caching for instant access to previously used objects.
- Build fast, powerful applications in record time!

Our price **$749** (SNEO)

STEPUP SOFTWARE

Guide Composer™ 1.2
by StepUp Software

- Create powerful Apple Guide help systems for any new or existing Macintosh application.
- Provides a WYSIWYG development environment: Guide content is developed in Guide windows.
- Design topics, phrases, and panels in the same format as the user will use them. Features are WYSIWYG interface, Topics, phrases, and hierarchical phrases, Coach marks, Fully-Integrated with Apple's Guide Maker (distributed with Guide Composer), compiles scripts automatically, PICTs in Panels, Generated Guide scripts are modifiable.
- FREE Update to all registered Guide Composer users. Demo is available at http://www.guideworks.com/.

Our Price **$99** (SGCOMP)

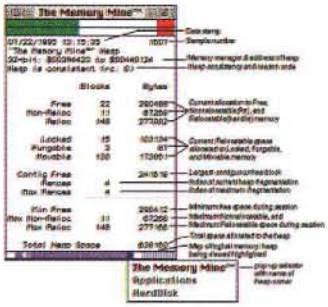
SEE RELATED PRODUCTS: AppleGuide Complete, Danny Goodman's AppleGuide Starter Kit, Real World AppleGuide

Memory Mine
by Adianta Inc.

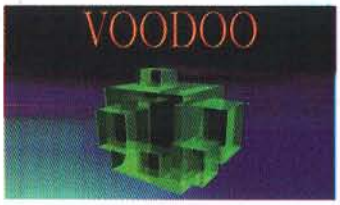
- Monitor heaps, identify problems such as memory leaks, and stress test applications
- Active status of memory in a heap is sampled on the fly: allocation in non-relocatable (Ptr), relocatable (Handle) and free space is shown, as are heap corruption, fragmentation, and more
- Allocate, Purge, Compact, and Zap memory lets users stress test all or part of a program

List $99 Our Price **$94.99** (SMEMMINE)

SEE RELATED CATEGORY: Dev. Environments



Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



VOODOO
by UniSoftware Plus

- Version control tool for the simple and clear management of projects in which files are created in numerous versions (variants and revisions).
- Allows both variant and revision control, and it manages not only variants and rev sions of single files, but of a whole software project (multi files, multi users, multi variants, access rights, etc.)
- Graphical user interface and is not only suitable for mere source code control but can handle all different kinds of files with amazing compression rates: typical size of delta between arbitrary files 5%
- Please note special prices for multiple copies:

Single license **$229** (SV00D001); 2 pack **$359** (SV00D002); 5 pack **$799** (SV00D005); 10 pack **$1369** (SV00D0010); 20 pack **$2399** (SV00D0020)

Additional pricing available on request.

SEE RELATED CATEGORY: Dev. Environments

QC High performance runtime stress testing for applications.
by Onyx Technology

- Tests include heap checks, purges, scrambles, handle/pointer validation, dispose/release checks, write to zero, de-reference zero as well as other tests like free memory invalidation and block bounds checking.
- Extremely user friendly – ideal for non-programmer testers.
- Also available in Japanese.

List $99 Our price **$94.99** (SQC)

One & 2-Day SHIPPING (call for details)

StoneTable
by StoneTablet Publishing

- Replaces all functions found in list manager
- Variable size columns/rows; different font, size, style, forecolor, backcolor per cell; sort, resize, move, copy, hide columns/rows; edit cells/titles in place; titles for columns/rows; multiple lines per cell; grid line pattern/color; greater than 32k data per table; up to 32k text per cell; support for balloon help and binary cell data. Versions for Think C, Think Pascal, MPW C, MPW Pascal, CodeWarrior 6 C.

StoneTable Extra
by StoneTablet Publishing

- Drag selected cells within table or to other tables.
- Add rows as part of drag; popup menus or check boxes in cells; variable width grid lines; move/drag/resize table in window; clipboard operations on multiple cells.
- Requires StoneTable.

This product can now be ordered in 3 different packages.

68K StoneTable
This includes StoneTable, StoneTableExtra for Think Pascal, Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$175** (SSTONE68)

PPC StoneTable
This includes StoneTable PPC, StoneTableExtra PPC for Think C, MPW C, MPW Pascal, CodeWarrior C, CodeWarrior Pascal

Our price **$175** (SSTONEPPC)

68K/PPC StoneTable
Includes both packages

Our price **$325** (SSTONEFAT)

SoftPolish CD-ROM by Bare Bones Software



- The essential tool for software quality assurance on the Macintosh
- Helps you identify inconsistencies with Apple's user interface guidelines, misspelled words, missing resources, and other mistakes
- Provides tools to put the finishing touches on software distribution packages prior to release
- Works independently of any programming language or environment
- Ideal for sanity checking software throughout the development process.

List price: $99 Our price: **$89** (SSOFTPOL)

PowerTap by Fortner Research

- Offers a simple method of implementing multi-processor capabilities to any application.
- Creates applications that are compatible with Apple's DayStar and other multi-processing computers.
- Compatible with Fortran, C, and C++ programming environments.
- Adjustable slide bar controls the level of CPU-sharing.
- Contains Thread Manager Options.
- Available now for MPW, CodeWarrior, Symantec Project Manager, PowerPC, A5, and A4.

Our Price **$299** (SPOWTAP)

PictureCDEF 1.3 by Paradigm Software

- Professional-level CDEF for creating custom graphical buttons (8-64 pixels) – used in products by Adobe, ProVue, STF Technologies and others!
- Multi-monitor and bit-depth sensitive.
- The button graphic (cicn, ResEdit) can be changed at runtime and even animated with a call-back routine.
- Create distinct buttons in seven variations: MultiState, PushButton, FlexiButton, ToggleButton, ChkButton, PushPictButton and TogglePictButton.
- Manual, sample code and MacApp 3.0 support included.

Full source code: **$95** (SPCDEF)

Object code: **$45** (SPICTOBJ)

dtF by dtF Americas

- True relational database system for Apple Macintosh computers.
- Provides a powerful choice for developers who want to create database centered applications with no performance trade-offs.
- Features SQL, full transaction control, error recovery, single user, client server architecture and multi-platform support including DOS, Windows, OS/2 and UNIX.
- The C/C++ API is identical and fully portable across all supported platforms.
- Third-party vendors supporting dtF will be able to offer a variety of advanced features and benefits to their customers royalty free.
- Tools are included for importing, exporting, creating and managing databases and users.
- Supported development environments include: Symantec, MPW, Metrowerks and more. Mac/SDK

List $695 Our Price **$679** (SDTF)

Spyer by InCider

- Easy to use tool that records all actions (including mouse movement) you perform on a Macintosh computer and then replays them at your preferred speed.
- Recorded data can be saved in files for future use.
- Works as a background process with any Macintosh application and is triggered by user defined Hot Keys.
- Enables the "Continuous Redo" utility and is especially useful for software testing and demonstration.

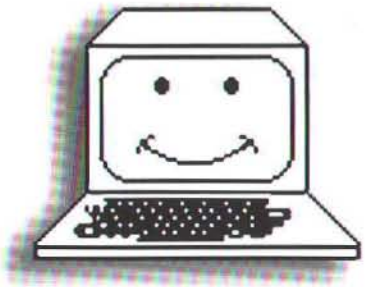
Our price **$39** (SSPY)

Spellswell Plus 2.0.4 by Working Software

- Award-winning, comprehensive, practical spelling checker that works in batch mode or within applications that incorporate the Apple Events Word Services protocol (e.g., Eudora, WordPerfect, Communicate!, and InfoDepot).
- Checks for spelling errors as well as common typos like capitalization errors, spaces before punctuation, double word errors, abbreviation errors, a/an before vowel/consonant, etc.
- MacTech orders include developer kit with Writeswell Jr., a sample AppleEvents Word Services word-processor and its source code.
- Available for OEM Sales.

Our price **$74.95** (SSPELL)

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



B-Tree HELPER 2.2

by Magreeable Software

- Inexpensive database engine for Macintosh programmers in C source code.
- Uses contiguous fixed length blocks.
- Expands the file as necessary and contracts files when possible.
- Inserts and deletes keys in one or more B-Trees.
- Finds keys equal to, less than, or greater than a given value in a few hundredths of a second.
- Finds lists of records whose keys are equal to, less than, or greater than a given value or are in a range of values.

Our Price **$150** (SBTREE)

Step-Up Installer Pack

by StepUp Software

- Package of several Installer "atoms" that let developers incorporate graphics, sounds, file compression and custom folder icons into installation scripts.
- Compression formats supported are Compact Pro & Diamond.
- Each atom also available separately.
- Compression requires additional licensing.

Our price **$219** (SINSTALL)

ScriptGen Pro

by StepUp Software

- Installer script generator which requires no programming or knowledge of Rez.
- Supports StepUp's InstallerPack, StuffIt decompression, Compact Pro decompression, custom packages, splash screens, network installs, and resource installation.

Our price **$169** (SSCRPTGEN)

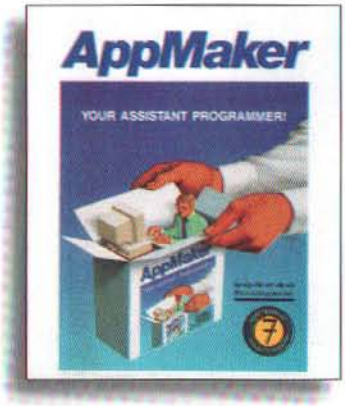


TestTrack-Bug Tracking the Macintosh Way

by Seapine Software, Inc.

- Tracks bugs, feature requests, test configurations, users, and more.
- Includes notifications, security, a powerful filter mechanism, and multiple reports.
- Links your testers, engineers, documentations staff, and project managers together to ensure all bugs are identified, fixed, and documented.
- Eliminates the need to build custom bug tracking solutions using general purpose database tools.
- Supports single- and multi-user bug databases (additional licenses required to use multi-user features).

Our price **$129** (STETR)



AppMaker

by Bowers Development

- Develop the user interface for a Macintosh application using the original interface builder.
- Just point and click to design your application.
- Creates resources and generates excellent source code.
- Supports most development environments including Metrowerks, Symantec, or MPW; C, C++, or Pascal; procedural or object-oriented, using PowerPlant, TCL, or MacApp.
- The generated code uses the Universal Headers to provide PowerMac compatibility.
- Great tool for beginners to learn object-oriented and Macintosh Toolbox programming techniques.
- Includes one-year subscription on CD and hardcopy documentation.

List $299 Our Price **$279.99** (SAPPMAKE)



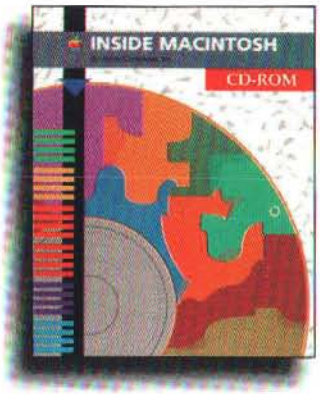
Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	LIST PRICE	OUR PRICE
3D Game Machine	S3DGAME	$299.00	$299.00
AdLib	SADLIB	195.00	195.00
Animation Class Library	SACL	250.00	250.00
CLImate	SCLIMATE	59.95	59.95
CMaster	SCMASTER	129.95	129.95
DataScript	SWDSCRIPT	249.00	229.99
ICONIX PowerTools-10 Pack	SICPP10	7,995.00	7,845.00
ICONIX PowerTools-6 Pack	SICPP6	5,995.00	5,945.00
ICONIX PowerTools-8 Pack	SICPP8	6,995.00	6,945.00
ICONIX PowerTools-AdaFlow	SICADA	1,495.00	1,395.00
ICONIX PowerTools-ASCII Bridge	SICASCII	1,495.00	1,395.00
ICONIX PowerTools-CoCoPro	SICCOCO	1,495.00	1,395.00
ICONIX PowerTools-DataModeler	SICDATAMOD	1,495.00	1,395.00
ICONIX PowerTools-FastTask	SICFASTTASK	1,495.00	1,395.00
ICONIX PowerTools-FreeFlow	SICFREEFL	1,495.00	1,395.00
ICONIX PowerTools-Object Modeler	SICOBJMOD	1,495.00	1,395.00
ICONIX PowerTools-PowerPDL	SICPOWER	1,495.00	1,395.00
ICONIX PowerTools-QuickChart	SICQUICKCH	1,495.00	1,395.00
ICONIX PowerTools-SmartChart	SICSMART	1,495.00	1,395.00
ICONIX Training & Consulting	TICONIX	3,000.00	2,945.00
IMSL Math and Stat F77	SIMSLSTAT	495.00	495.00
Info-Mac VIII	SINFOMAC8	39.95	35.95
LJ Profiler	SLJPROF	295.00	295.00
Mac Games II	SMACGAMES2	29.95	26.95
Mac Source	SMACSOURCE	29.95	26.95
MacA&D Demo	SMACADD	149.00	75.00
MacA&D Product	SMACADP	2,995.00	1,995.00
MacAnalyst Demo	SMACAND	79.00	74.95
MacAnalyst Product	SMACANP	995.00	945.00
MacAnalyst/Expert Demo	SMACANED	79.00	74.95
MacAnalyst/Expert Product	SMACANEP	1,595.00	1,545.00
MacDesigner Demo	SMACDESD	79.00	74.95
MacDesigner Product	SMACDESP	995.00	945.00
MacDesigner/Expert Demo	SMACDESED	79.00	74.95
MacDesigner/Expert Product	SMACDESEP	1,595.00	1,545.00
MacScholar	SMACSCHOL	29.95	26.95
MacScholar Jr.	SMACSCHOLJR	29.95	26.95
MacWireFrame	SFRAM	299.00	75.00
MAScript	SMASCRIPT	199.00	195.00
More Savvy	SMORSAV	450.00	445.00
PowerMac 2	SPOWERMAC2	29.95	26.95
Professional MachTen for 68k Macs	SPROM10	695.00	695.00
Rosanne	SROSANNE	595.00	595.00
S-Case	SSCASE	495.00	395.00
Savvy	SSAVVY	250.00	245.00
Savvy QuickTime	SSAVVYQT	250.00	245.00
StoneTable for 68K	SSTONE68	175.00	175.00
StoneTable for PowerPC	SSTONEPPC	175.00	175.00
Super Savvy	SSSAVVY	700.00	695.00
SuperPlot	SSPLOT	195.00	145.00
SuperPlot Pro	SSPLOTPRO	349.00	349.00
Tenon Ported Applications CD	SPORTED	50.00	49.95
Top 10 Pick for Macintosh	STOP10PICK	49.95	44.95
V3d/3dPane/SmartPane	SQ3	192.00	192.00
VText	SVTEXT	350.00	350.00

INSIDE MACINTOSH

by Apple Computer, Inc.

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



Inside Macintosh: CD-ROM by Apple Computer, Inc.

- More than 25 volumes in electronic form.
- Includes: QuickDraw™ GX Library, Macintosh Human Interface Guidelines, PowerPC System Software, Macintosh Toolbox Essentials and More Macintosh Toolbox, QuickTime and QuickTime Components.
- Access over 16,000 pages of information with Hypertext linking and extensive cross referencing.

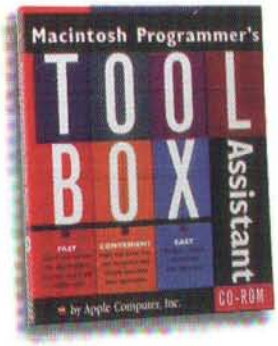
List $99.95 Our Price **$89.95** (BIMCD)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

WAIT... There's More!

Here are all of the Inside Macintosh products – half off! For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us.

PRODUCT	CODE	LIST PRICE	OUR PRICE
Inside Macintosh: AOCE Applications Interface	BIMAOCE	$44.95	$21.99
Inside Macintosh: AOCE Service Module	BIMAOCES	29.95	14.50
Inside Macintosh: Devices	BIMDEV	29.95	14.50
Inside Macintosh: Files	BIMFIL	29.95	14.50
Inside Macintosh: Imaging with QuickDraw	BIMIMAG	32.95	15.99
Inside Macintosh: Interapplication Communications	BIMIAPP	36.95	17.99
Inside Macintosh: Macintosh Toolbox Essentials	BIMTBOX	39.95	17.45
Inside Macintosh: Memory	BIMMEM	24.95	11.99
Inside Macintosh: More Macintosh Toolbox	BIMMAC	34.95	16.99
Inside Macintosh: Networking	BIMNET	29.95	14.50
Inside Macintosh: Operating System Utilities	BIMOPSU	28.95	13.99
Inside Macintosh: Overview	BIMOVER	24.95	11.99
Inside Macintosh: PowerPC Numerics	BIMPPCNUM	28.95	13.99
Inside Macintosh: PowerPC System Software	BIMPPCSYS	24.95	11.45
Inside Macintosh: Processes	BIMPROC	22.95	10.99
Inside Macintosh: QuickDraw GX Environ. & Utilities	BIMGXENV	31.95	15.50
Inside Macintosh: QuickDraw GX Graphics	BIMGXGR	31.95	15.50
Inside Macintosh: QuickDraw GX Objects	BIMGXOBJ	31.95	15.50
Inside Macintosh: QuickDraw GX Printing	BIMGXPRNT	29.95	14.50
Inside Macintosh: QuickDraw GX Printing Extensions	BIMGXEXT	29.95	14.99
Inside Macintosh: QuickDraw GX Prog. Overview	BIMGXOV	24.95	11.99
Inside Macintosh: QuickDraw GX Typography	BIMGXTYP	29.95	14.50
Inside Macintosh: QuickTime	BIMQT	29.95	14.50
Inside Macintosh: QuickTime Components	BIMQTCOM	34.95	15.95
Inside Macintosh: Sound	BIMSOUND	29.95	14.50
Inside Macintosh: Text	BIMTEXT	39.95	19.50
Inside Macintosh: X-Reference	BIMXREF	19.95	9.50



Programmer's Toolbox Assistant CD-ROM

Instant electronic access to Inside Macintosh essentials.
by Addison-Wesley Publishing

- Get quick access to reference pages for over 4,000 Toolbox calls in your system software from their development environment.
- Essential information for Macintosh software developers.
- Hypertext links allow programmers to view related topics easily.
- The ultimate electronic reference tool for Macintosh programmers.

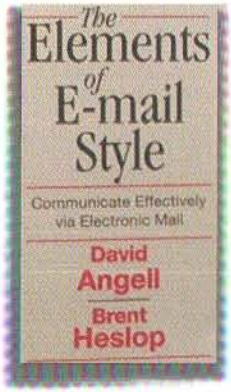
List $99.95 Our Price **$89.95** (STBASST)

AppleScript Applications: Building Applications with FaceSpan and AppleScript

by John Schettino Affiliation & Liz O'Hara

• Build complete AppleScript applications using FaceSpan, a user interface development tool that makes AppleScript applications truly "Mac-Like" • Uses a step-by step approach to demonstrate techniques for building applications through illustrations and samples • Provides Graphical User Interface (GUI) design tips and practical approaches for implementation • Contains one CD-Rom with AppleScript 1.1, a demonstrations version of FaceSpan 2.1, source code for all example applications numerous AppleScript shareware and demonstrations programs • Contains a section on debugging AppleScript applications using FaceSpan

List $34.95 Our Price **$31** (BAPSCAP)

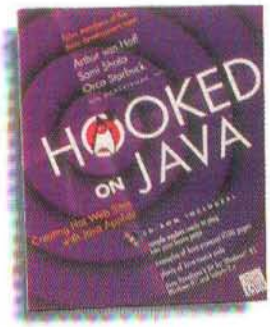


The Elements of E-Mail Style

by Brent Heslop and David Angell

- Write solid, effective E-Mail and avoid common pitfalls.

List $14.95 Our Price **$13.45** (BEMAIL)



Hooked on Java

by Arthur Van Hoff, Sami Shaio, and Orca Starbuck

- Written by the Java development team at Sun Microsystems, Inc.
- An introduction to using applets, for Web administrators, designers, and developers.
- Demonstrates how to use applets in your own pages.
- Includes a concise introduction to the Java language, and a CD with tools.

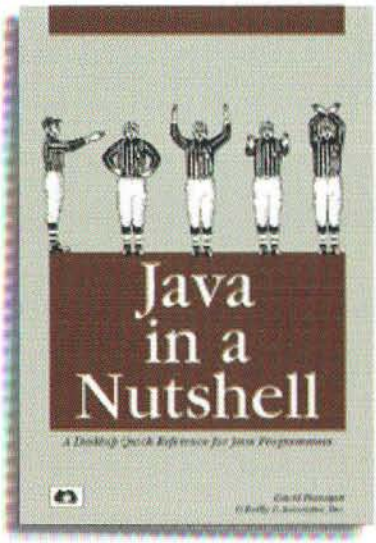
List $29.95 Our Price **$26.95** (BHJAVA)

Java in a Nutshell

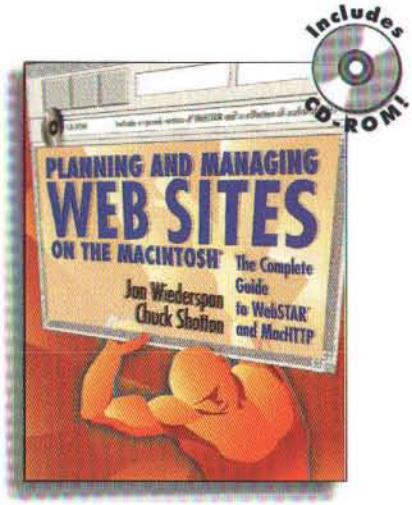
by David Flanagan

- A complete quick reference guide to Java, the hot new programming language from SunMicrosystems.
- Contains descriptions of all of the classes in the Java 1.0 API, with a definitive listing of all methods and variables.
- Also contains an accelerated introduction to Java for C and C++ programmers who want to learn the language fast.

List $14.95 Our price **$13.45** (BJAVANUT)



Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



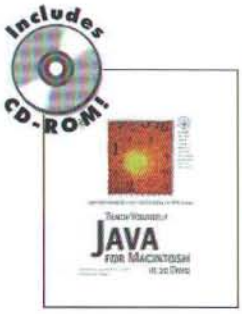
Planning and Managing Websites

by Jon Weiderspan and Chuck Shotton

- The definitive guide to setting up and running a Web site on the Macintosh.
- Learn everything you need to know about using WebSTAR, the best known HTTP server software and its shareware predecessor MacHTTP.
- Write CGI applications for your server – in AppleScript and in C.
- CD includes a special version of WebSTAR, plus tons of useful software.

List $39.95 Our Price **35.96** (BPLANWEB)

Best Deals All Prices Reduced 10% On Books!



Teach Yourself Java for Macintosh in 21 Days

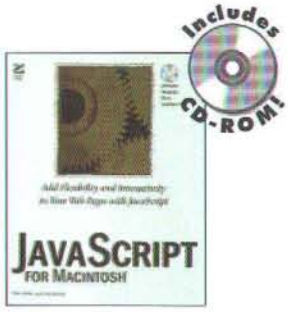
by Laura Lemay and Charles L. Perkins with Timothy Webster

- Add interactivity and multimedia to Web pages!
- A step-by-step guide to make your Web site come alive.
- Learn the basics of programming Java applets and the concepts behind the Java language.
- Includes CD-ROM with a limited version of Roaster, the first commercial, integrated applet development environment for Java for the Macintosh!

List $40 Our price **$36** (BJAVAMAC)

JavaScript for the Macintosh

by Matt Shobe and Tim Ritchey



- Allows non-programmers to take advantage of the power of Netscape Navigator.
- Expand the capabilities of your Web page, without having to understand C or C++.
- CD-ROM contains "Wizlets" that allows you to easily create your own JavaScripts
- Takes you step-by-step through programming cross-platform JavaScripts
- Details how to create JavaScripts for JavaScript-aware Web browsers

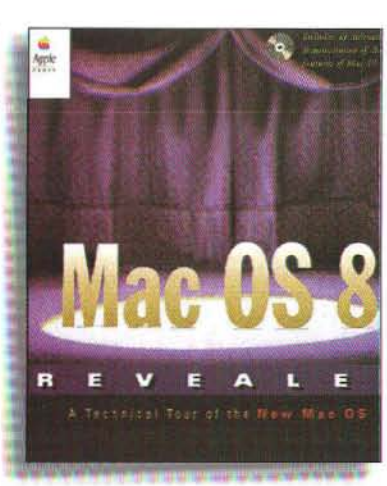
List $45 Our price **$40.50** (BJAVASCRPTJ)

Mac OS 8 Revealed

by Tony Francis

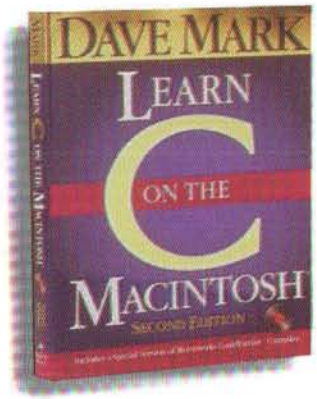
- The first authoritative look at this exciting new operating system.
- A must for Mac developers who want to make their software compatible with Mac OS 8.
- Essential for system administrators who plan to upgrade their system.
- Included CD-ROM contains demos of new Mac OS 8 features.

List: $34.95 Our Price: **$31.45** (BMACOS8R)



Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

1-800-MACDEV-1 • Outside U.S. & Canada: 805-494-9797 • Fax: 805-494-9798



Learn C on The Macintosh Second Edition

By Dave Mark

- New revised edition.
- Easy-to-understand – everything you need to start programming!
- Updated and enhanced exercises that lead you step by step. You'll learn function, variables, pointers datatypes, data structures, file input and output and more!
- Includes CD-ROM with Metrowerks CodeWarrior™ Lite – the hottest Macintosh programming environment (including a PowerPC version).

List $34.95 Our Price **$31.45** (BLEARNC2)

CodeWarrior Software Development Using PowerPlant

by Jan L. Harrington

- C++ programmers will learn to develop object-oriented software applications for the Mac and Power Mac using the PowerPlant environment and the classes that support it.
- Covers CodeWarrior 8. • Included CD-ROM contains source code for all the programming examples in the book and Metrowerks CodeWarrior Lite.

List Price: $34.95 Our Price: **$31.45** (BCWSWDEV)

One & 2-Day SHIPPING (call for details)

Designing Animation for the Web

by Hayden Development Team

- Provides technical information and design advice for creating animation on the web
- Expert tips show which technology to use for specific needs: GIF animation, Shockwave, Java and more.
- Covers bandwidth considerations and the integration of animations into overall site design.
- Step-by-step instructions.
- CD-ROM includes tools, scripts, and examples for creating animations.

List $40 Our Price **$36** (BDAFTW)

CGI By Example

by Jeffrey Dwight

- CD-ROM contains all the code and tools used in the book, and additional tools and examples, as well as related chapters from Special Edition Using CGI and Special Edition JavaScript
- Includes end of the chapter review questions and exercises to reinforce the learning process
- Covers basic CGI applications, as well as advanced topics like server administration issues and database connectivity

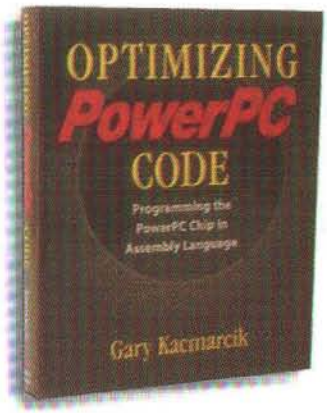
List $34.99 Our Price **$31.45** (SSGIBE)

Perl Quick Reference

by Michael O. Foghlu

- Commands are sorted by task, class packet, platform or hardware compatability all in alphabetical order
- Includes jump tables to guide reader to specific pages in the reference
- Designed in a larger trim size than traditional Quick References-and with a lay-flat binding

List $19.99 Our Price **$17.99** (BPERLREF)



Optimizing PowerPC Code: Programming the PowerPC in Assembly Language

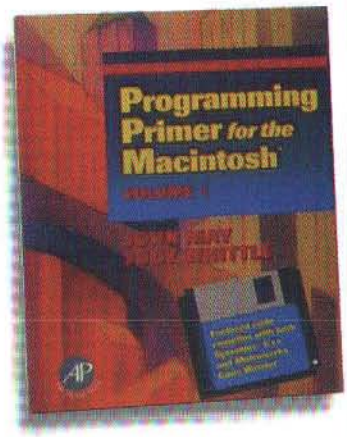
by Gary Kacmarcik

- Take full advantage of the potential of the PowerPC by mastering the Assembly Language techniques.
- Learn to produce faster more robust software!

List $39.95 Our Price **$35.96** (BOPTPPC)

Macintosh C Programming Primer Volume I

Second Edition, Inside the Toolbox Using THINK C by Dave Mark and Cartwright Reed



- Updated new edition of the Macintosh programming best seller.
- System 7, new versions of THINK C and ResEdit.
- Learn how to use the resources, Macintosh Toolbox and interface to create stand-alone application
- 672 pages.

List $26.95 Our Price **$24.25** (BCPRIM1)

Macintosh C Programming Primer Volume II

Mastering the Toolbox Using THINK C by Dave Mark

- Covers advanced topics such as: Color QuickDraw, THINK Class Library, TextEdit, and the Memory Manager: 528 pgs.

List $26.95 Our Price **$24.25** (BCPRIM2)



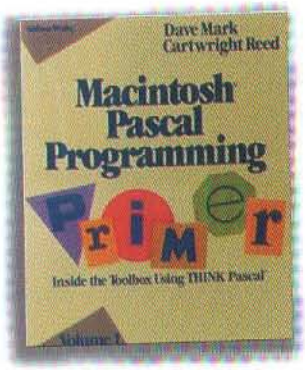
Learn C++ on the Macintosh

by Dave Mark

- Basic syntax of C++ and object programming.
- Learn how to write, edit, and compile your first C++ programs.
- Features key C++ concepts such as derived classes, operator overloading, iostream functions and more.
- Includes a special version of Symantec C++ for Macintosh. Book/disk package with 3.5" 800K Macintosh disk. 400 pages.

List $36.95 Our Price **$33.26** (BLRNCPP)

SEE RELATED CATEGORY: Dev. Environments

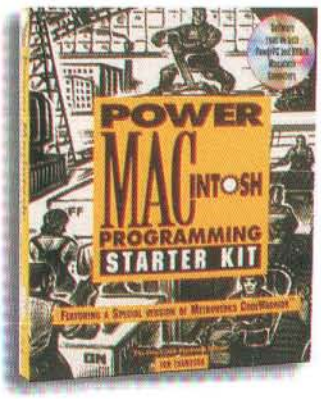


Macintosh Pascal Programming Primer Volume I

Inside the Toolbox Using THINK Pascal
by Dave Mark and Cartwright Reed

This tutorial shows programmers new to the Macintosh how to use the Toolbox resources, and the Macintosh interface to create stand-alone applications with Symantec's THINK Pascal. 544 pages.

List $26.95 Our Price **$24.25** (BPASCPRI)



Power Macintosh Programming Starter Kit

by Tom Thompson

- Enter the world of the PowerPC chips.
- Get the scoop on the microprocessors, the RISC architecture, and how to write native code and emulation operations to create software for the Macintosh PowerPC.
- CD-ROM includes a unique compiler for writing code easily.

List $39.95 Our Price **$35.10** (BPPCSTART)

SEE RELATED CATEGORY: Dev. Environments



Macintosh Programming Secrets 2nd edition

By Scott Knaster and Keith Rollin

- Macintosh Programming Secrets is divided in two parts.

Part 1: "Concepts and Ideas", discusses the evolution of the Macintosh and the standards, customs, and software that shape the system as well as the Macintosh user interface.
Part 2: "Technical Adventures", presents the skeleton of an application, and then builds upon that framework to describe how to: • Create fancy dialogue boxes • Utilize the new 32 bit QuickDraw developments • Track the mouse with "marching ants" • Manage multiple windows with the Window manager • Copy files within a program • Install the worlds strangest spinning cursor.

List $31.95 Our Price **$28.76** (BPSECRET)

Order Toll-free
800-MACDEV-1
(800-622-3381)

Cyberdog Programmers Kit

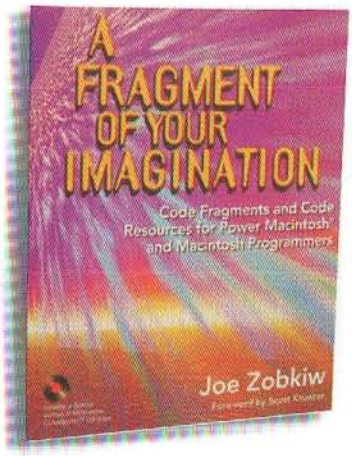
by Apple Computer, Inc.

Includes CD-ROM!



- Apple's official reference.
- A must for developers who want to make customizable Internet access available to all Mac users.
- Included CD-ROM contains all the tools needed to create Cyberdog-aware components.

List: $39.95 Our Price: **$34.95** (BCYBERDOG)



A Fragment of Your Imagination

by Joe Zobkiw

Includes CD-ROM!

- Packed with useful code fragments for the Macintosh and Power Macintosh.
- Hard to find information about techniques used to structure and build fat, safe fat, and accelerated code resources.
- All code is reusable and is provided on the disc, along with Metrowerks Code Warrior Lite. Book/CD-ROM, 528 pages.

List $39.95 Our Price **$35.96** (BFRAG)

Programming with AppleTalk

by Michael Pierce

- Programming with AppleTalk is the hands-on guide to understanding and working with AppleTalk. Topics covered include:
- How to create applications and system extensions that run with AppleTalk.
- AppleTalk protocols and the protocol stack, transport media, the Preferred AppleTalk Intefrace, and the storage management.
- Numerous working code examples walk you through using RDEV, INIT, NBP, ATP, and ADSP. You will also learn the use of: Synchronous, and asynchronous calls, How to avoid heap fragmentation, And how to configure a Chooser Interface.

List $24.95 Our Price **$22.45** (BPROAT)

Java Language API SuperBible

by Daniel Groner, Todd Sundtsed, Casey Hopson, Harish Prabandham

- Covers Java 1.1
- Hundreds of concrete source code examples provided
- Sample projects introduced and assembled in each chapter

List $59.99 Our price **$53.99** (BJLAS)

NetObjects Fusion Handbook

from Hayden Development Team

- Create Web pages and Sites through design techniques developed by Clement Mok's design agency.
- Step-by-step instructions on updating sites with links and navigational controls automatically.
- Outlines the comprehensive steps to delivering an automated, turnkey Web publishing solution.
- CD-ROM contains a 45 day time-limited version of NetObjects Fusion.

List $50 Our Price **$45** (BNETOFH)

Lowest price guaranteed. Feel Good about your purchase! 30-day return policy.

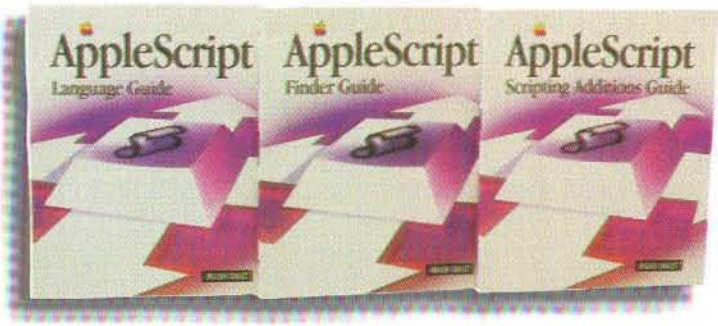
Infini-D Revealed

by Brendan Donahoe & Adam Lavine

- CD-ROM includes a demo version of Infini-D 3.1, files for working through the tutorials, and a gallery of spectacular 3D images
- Shows how to use the new animation features to bring images to life
- Details Infini-D's new and improved color-coded interface

List $45 Our price **$40.50** (SINFDREV)

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



AppleScript Language Guide,
by Apple Computer, Inc.

- A complete reference for anyone using AppleScript to modify existing scripts or to write new ones.
- Contains useful information for programmers who are working on scriptable applications or complex scripts.
- Features detailed definitions of AppleScript terminology and syntax in the following categories: Value classes, commands, objects and references to objects, expressions, control statements, handlers, and script objects.
- Includes many sample scripts, discusses advanced topics such as writing command handlers for script applications, the scope of script variables and properties declared at different levels in a script, and inheritance and delegation among script objects.

List $29.95 Our Price **$26.95** (BALG)
SEE RELATED CATEGORY: Scripting

AppleScript Finder Guide, English Dialect
by Apple Computer, Inc.

- Provides definitions for Finder object classes and commands.
- Write, record, or run scripts that trigger the same desktop actions that you trigger using the keyboard and mouse.

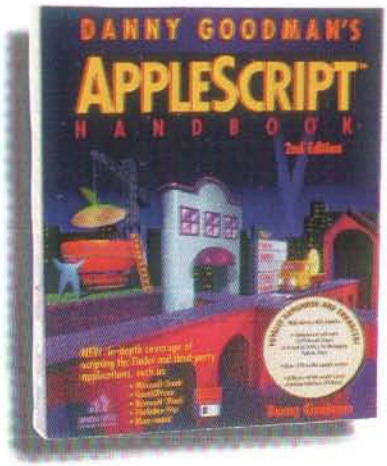
List $19.95 Our Price **$17.95** (BAFG)
SEE RELATED CATEGORY: Scripting

AppleScript Scripting Additions Guide
by Apple Computer, Inc.

- Use the standard scripting additions commands.
- Write scripting additions.

List $18.95 Our Price **$17.05** (BSCRADD)
SEE RELATED CATEGORY: Scripting

Best Deals On Books! All Prices Reduced 10%



Danny Goodman's AppleScript Handbook Second Edition by Danny Goodman

Includes Disk!

- Customize and extend the capabilities of any Macintosh computer – no programming experience needed!
- Learn to use scripts to enhance the Macintosh environment, automate many processes, link data between applications, and much more.
- All-new examples showing how to integrate AppleScript with the Finder, spreadsheets, desktop publishing programs, graphics applications, databases, telecommunications programs, utilities, and HyperCard.
- Includes 3½" disk with over $100 worth of software, including AppleScript 1.1, valuable utilities, and powerful, ready-to-use scripts.

List $39 Our Price **$35** (BDGASHB)
SEE RELATED CATEGORY: Scripting, Tools, Libs & Utilities

Applied Mac Scripting
by Tom Trinko

- Learn to design and develop powerful scripts.
- Covers AppleScript™, Frontier, QuicKeys, Tempo II, nShell, FaceSpan Application Builder, Scripting PlainTalk and System 7.5.
- Hands on tutorial shows you how to automate your Macintosh activities by learning how to use the AppleScript and Frontier scripting environments.
- Harness the capabilities of a wide variety of Macintosh applications into the integrated productivity tools. This includes such things as the newspaper script which combines the power of SITcomm, MacWrite Pro, and FileMaker Pro, or QuarkXPress.

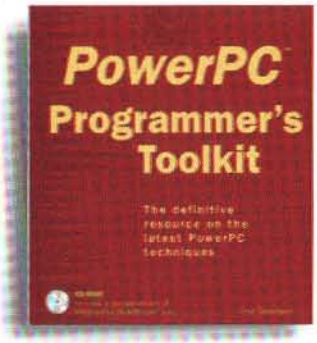
List $34.95 Our Price **$31.45** (BAPPLIED)
SEE RELATED CATEGORY: Scripting

PowerPC Programmer's Toolkit

by Tom Thompson

- CD-ROM includes a special version of Metrowerks CodeWarrior 7.0 and sample code from the book
- Details how to write PowerPC applications in native code for blazing speed
- Written by an Apple Insider

List $45 Our Price **$40.50** (SPPCPT)



Mastering Netscape 2.0 for Macintosh, Second Edition by Greg Holden

- CD-ROM includes Netscape Plug-Ins and helper applications
- Uncovers hidden features of the program and how to best utilize them
- Web page for the book includes cool utilities and updated information

List $40 Our Price **$36** (SMASNET4)

Network Frontiers Bundle

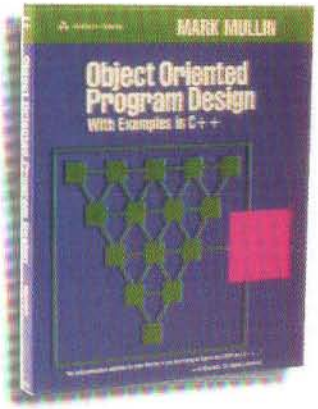
by AP PROFESSIONAL

(Includes 3 books: Complete Guide to MAC,Backup Management, Designing AppleTalk Network Architectures, Managing AppleShare and Workgroup Servers)

- Apple Certified
- Each bundle includes the first 3 books that correspond to the Apple
- Certified Server Engineer (ACSE) program

List $89.95 Our Price **$59.95** (BNETFB)

One & 2-Day SHIPPING (call for details)



Object Oriented Program Design

by Mark Mullin

- A concise guide to the essential concepts and techniques of OOP design
- Clarifies the key concepts of object oriented programming such as objects, classes, entities, hierarchies, and inheritance
- Uses typical database application to illustrate each OOP topic, to give the programmer a familiar point of reference

List $22.95 Our Price **$20.66** (BOOPRODES)

Intranet Web Development: *Enterprise Alternatives to Client/Server Computing* by Hayden Development Team

- Learn why traditional client/server computing is being replaced by new WWW technologies.
- Discusses WWW application development with the Microsoft Visual Script enterprise application developer in mind.
- Learn techniques for the conversion of existing Visual Basic, C++ and PowerBuilder Applications to the new WWW platform.

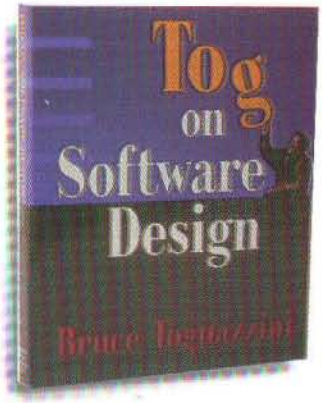
List $49.99 Our Price **$45** (BINTWD)

DeBabelizer: The Authorized Edition

by Hayden Development Team

- Teaches effective ways to utilize DeBabelizer to improve graphic's quality and speed.
- Offers techniques for using SuperPallette to optimize images.
- Provides extensive visual examples to illustrate key production techniques in all areas of graphics processing.
- CD-ROM includes the Lite version as well as Photoshop plug-ins.

List $45 Our Price **$40.50** (BDEBTAE)



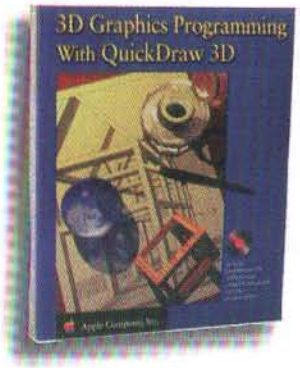
Tog on Software Design

by Bruce "Tog" Tognazzini

Respected industry futurist, Tog, presents his vision of our technological future, detailing the steps computer professionals need to take to deliver new technologies that will profit the industry and benefit society in general. Contains Tog's insights on a wide range of topics from quality management to the meaning of standards, and responses to queries supplied by designers and developers.

List $29.95 Our Price **$26.95** (BTOG)

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



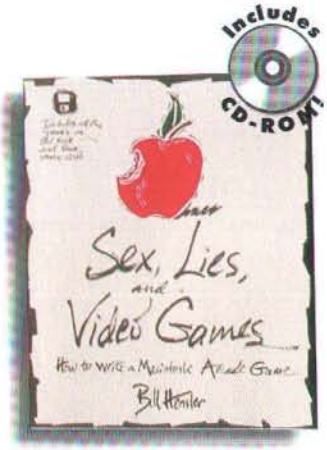
3D Graphics Programming Using QuickDraw 3D

Includes CD-ROM!

by Apple Computer, Inc.

- Incorporate spectacular 3D graphics into your applications.
- Explore QuickDraw 3D, a revolutionary graphics extension to the Mac OS for Power Macintoshes.
- CD contains the complete QuickDraw 3D system itself and a complete database of the QuickDraw 3D API, allowing you instant access to the hundreds of graphics calls via a fast viewing engine. Book/CD-ROM, 640 pages.

List $39.95 Our Price **$35.96** (B3DGRAP)



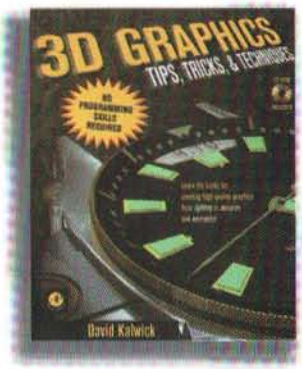
Sex, Lies and Video Games

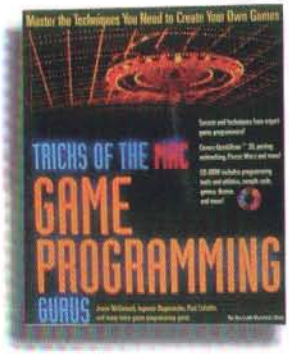
by Bill Hensler

- A learn-by-example tutorial on the ins and outs of Mac arcade-style game programming in C.
- Features game theory, sprite animation, sound, and interaction techniques.
- A must-read for serious programmer's and hobbyists alike.

List 34.95 Our Price **$31.46** (BSEX)

SEE RELATED CATEGORY: Tools, Libs & Utilities





Tricks of The Mac Game Programming Gurus

by McCornack, Ragnemalm, Celestin, et al.

- For beginning to expert game programmers
- Complete overview of all the necessary components of game programming on the Macintosh.
- Packed with valuable tools, utilities, sample code, CodeWarrior™ Lite and game demos.
- QuickDraw 3D and Power Mac optimization and inside info on how Glypha III was created.
- Hundreds of tried-and-true tricks, tips, and insider secrets from well-known Mac game programming experts

List $50 Our Price **$45** (BTRICKS)

3D Graphics-Tips, Tricks, & Techniques

by David Kalwick

- Written from a user's point-of-view, this book covers all the important 3D techniques including lighting, textures, animation, camera angles, and perspectives.
- Through easy-to-understand examples you learn how to create high-quality 3D graphics.
- Includes a Windows/Macintosh CD-ROM featuring tutorials, 3D examples from the book, and more.

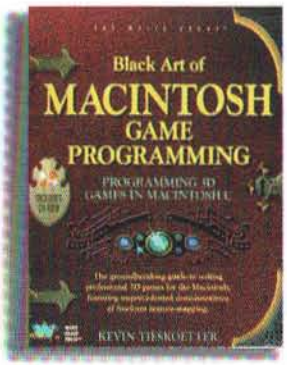
List 34.95 Our Price **$31.46** (B3DGTTT)

Black Art of Macintosh Game Programming

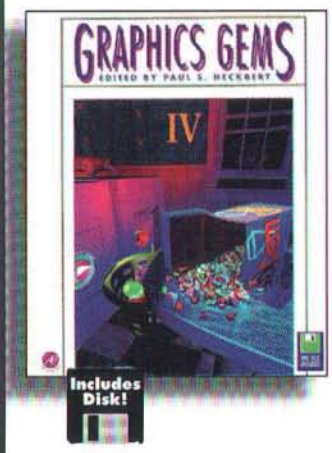
by Kevin Tieskoetter

- Develop your own 3D games in C on the Mac.
- Includes CD with project files for both Symantec C and Code Warrior
- Create freeform texture-mapped games and polygon graphics
- Control dynamic source code-all compatible as native to the Power Mac
- Write directly to the screen, bypassing QuickDraw

List $39.99 Our price **$35.99** (BBLACK)



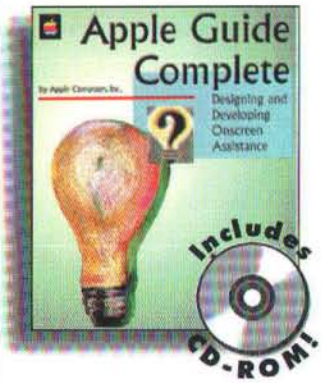
Includes CD-ROM!



Graphic Gems V Edited by Alan W. Paeth

- Loaded with practical tools for implementing new ideas and techniques, to offer working solutions to real programming problems.
- Contains over 40 new gems in ellipses, splines, Bezier curves, and ray tracing – displaying the most recent and innovative techniques in graphics programming.
- Includes a disk with source code from all five volumes. Available in both IBM and Macintosh versions. CONTENTS: Algebra and Arithmetic. Computational Geometry. Modeling and Transformation. Curves and Surfaces. Ray Tracing and Radiosity. Halftoning and Image Processing. Utilities.

List $49.95 Our Price **$44.95** (BGEMS5)

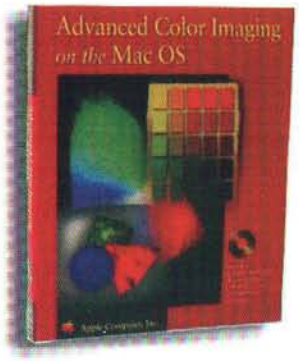


AppleGuide Complete
by Apple Computer, Inc.

- Covers Guide Maker, the software you use to build and test guide files.
- Learn about the complete cycle of designing as well as advanced topics such as scripting and coding guide files. Book/CD-ROM, 544 pages.

List $39.95 Our Price **$35.96** (BAPLGD)

SEE RELATED CATEGORY: Tools, Libs & Utilities



Advanced Color Imaging on the Mac OS
by Apple Computer, Inc.

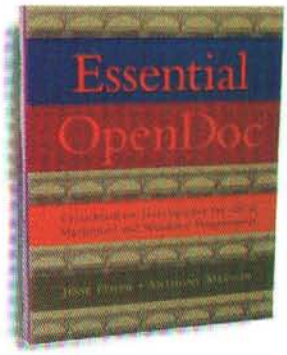
- Enhance your software's color capabilities with step-by-step instructions.
- Augment the color support supplied with QuickDraw, and QuickDraw GX.
- Use the Pallette Manager to get the best colors on limited displays.
- Match colors between screens and input/output devices (scanners & printers)
- CD includes a complete reference information in both QuickView and Acrobat formats. Plus, a sample application demonstrating ColorSync programming techniques.

List $36.95 Our Price **$33.25** (BADVCI)

OpenDoc Programmer's Cookbook by Apple Computer, Inc.

- Shows you how to create OpenDoc software components, called parts editors, for the Mac OS Platform.
- Including instructions for setting up the Macintosh Programmers Workshop (MPW) development environment to write OpenDoc software
- Annotated listings of explaining the methods that implement the SamplePart part editor
- Descriptions of other sample part editors created by the OpenDoc engineering team to illustrate more advanced features
- Summary descriptions of software utilities provided with OpenDoc for the Mac OS
- An Introduction to the System Object Model (SOM) technology underlying OpenDoc

List $24.95 Our price **$22.45** (BODCOOK)

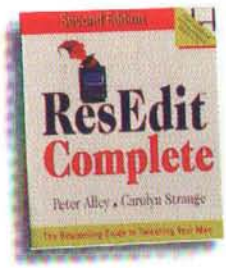


Inside PowerPlant Manual
by Metrowerks

- Create PowerPlant applications using the CodeWarrior IDE and PowerPlant Constructor.
- Full descriptions of major PowerPlant classes and resources.
- Included are the PowerPlant Constructor Manual, including View, TextTraits and Custom Types editing, and PowerPlant Library Reference, covering all classes and functions in PowerPlant.

Our Price **$34.95** (BINSPP)

SEE RELATED CATEGORY: Dev. Environment

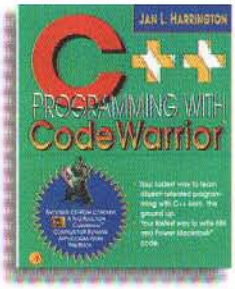


ResEdit Complete, Second Edition
by Peter Alley and Carolyn Strange

- Customize every aspect of your interface form creating screen backgrounds and icons to customizing menus and dialog boxes. 608 pages. Book/disk package.

List $34.95 Our Price **$31.45** (BRESED2)

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



C++ Programming with CodeWarrior

by Jan L. Harrington

- Beginning OOP for the Macintosh and Power Macintosh and Mac OS compatibles.
- Learn object-oriented programming techniques using C++ as the example language and Metrowerks and CodeWarrior as the example compiler.
- Enclosed CD contains example code from the book and a full-function Metrowerks CodeWarrior

List $35.95 Our Price **$32.35** (BCPPCW)

Includes CD-ROM!

OpenDoc Programmer's Guide

by Apple Computer, Inc.

• The official reference for the implementation of OpenDoc on the Mac OS. • Describes the component software revolution and explains how to develop for it on the Mac OS platform. • Accompanying CD-ROM contains a complete reference to the OpenDoc programming interface, and an extensive collection of tested, reusable sample code.

List $44.95 Our Price **$40.46** (BOPENDOC)

The ResEdit All Night Diner

by David Ciskowski

Best Deals On Books! All Prices Reduced 10%

- An idea-filled menu and introduction to the joys of customizing software.
- Add personality to the Mac by customizing default icons, the text of menus and dialog boxes, cursors, pointers and more.
- Disk features ResEdit, plus lots of sample resources

Includes Disk!

List $24.95 Our Price **$22.45** (BRESDINE)

SEE RELATED CATEGORY: Dev. Environments

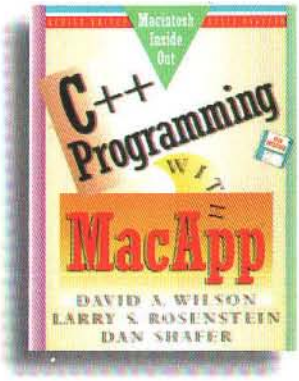
C++ Programming with MacApp

by David Wilson, Larry Rosenstein & Dan Shafer

- Learn the secrets to unlocking the power of MacApp®, Apple's development environment for C++
- Learn to design complex windows and views using the ViewEdit tool
- Learn to support multipage text and graphics with only five lines of code
- Learn to support Undo for menu commands and drawing operations that use the mouse.

List $34.95 Our Price **$31.46** (BCPPMACAP)

SEE RELATED CATEGORY: Tools, Libs & Utilities



Essential OpenDoc

by Jesse Feiler and Anthony Meadow

- Gives an in-depth look at the technical issues of OpenDoc
- Explores the three core technologies that support it's functionality – SOM, OpenDoc's storage mechanism, and the Open Scripting Architecture (OSA).
- Also examines CyberDog, a set of OpenDoc part editors that provides access to Internet services and offers compelling example of the power of OpenDoc development

List $39.95 Our price **$35.95** (BESOD)

Inside CodeWarrior 10

by Metrowerks

Includes CD-ROM!

- Includes CodeWarrior IDE User's Guide.
- This is the printed version of the documentation provided on the CD.
- Covers CodeWarrior, the debugger, and associated tools.

Our Price **$34.95** (BINSCW)

SEE RELATED CATEGORY: Dev. Environment

Metrowerks CodeWarrior Programming

by Dan Parks Sydow

- Includes CodeWarrior Lite, and Full Coverage of PowerPlant™.
- The best information on Metrowerks CodeWarrior, giving full coverage to the Gold Edition.
- CD includes Code Warrior Lite.

List $39.95 Our Price **$35.95** (BCWPROG)

Order Toll-free
800-MACDEV-1
(800-622-3381)

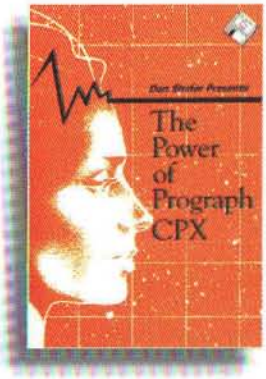
Visual Programming with Prograph CPX

by Scott B. Steinman and Kevin G. Carver

- An introduction to the language and a guide for advanced users, for both Macintosh and Windows-based machines.

List $34 Our Price **$30.60** (BVISPRO)

SEE RELATED CATEGORY: Dev. Environments

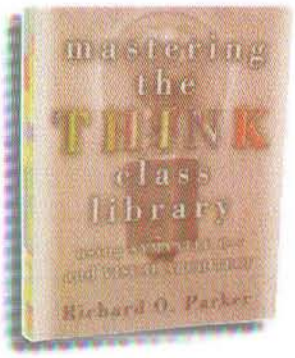


The Power of Prograph CPX

by Dan Shafer

- Master the revolutionary graphical object-oriented programming language.
- Step by step course through three interrelated projects of increasing complexity.
- Learn Prograph language, CPX classes and object editors.
- Includes disk with all code in the book.

List $49.95 Our Price **$19.95** (BDANPRO)



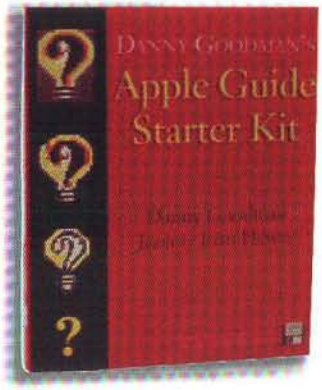
Mastering the THINK Class Library

by Richard Parker

- Provides a thorough examination of Symantec's extensive Class Library and the Visual Architect.
- A complete description of the structure and operation of the TCL includes explanations of all code generated by the Visual Architect, any necessary custom code, and the operation of this code.
- Visual Architect tutorials provide you with a step-by-step approach for simplifying the development of complex Macintosh applications. 496 pages.

List $29.95 Our Price **$26.95** (BMASTERTCL)

SEE RELATED CATEGORY: Dev. Environments



Danny Goodman's Apple Guide Starter Kit

by Danny Goodman and Jeremy Joan Hewes

- Create your own Apple Guide databases quickly and easily, without having to learn a scripting language, write coded files, or use several different files and programs
- Includes advice and tips on how to design a good Guide, from planning and creation through testing, revising, and indexing. Book/disk, 320 pages.

List $34.95 Our Price **$31.46** (BDGAGSK)

MacsBug Reference & Debugging Guide For MacsBug version 6.2

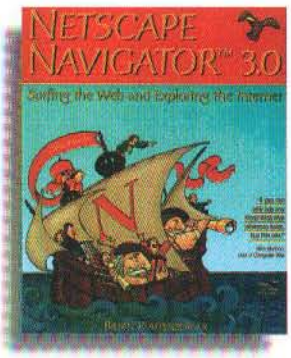
by Apple Computer, Inc.

• MacsBug is an assembly-language-level debugging tool • Macros, templates, dcmds, and other resources for making debugging easier • Macintosh memory management and the operating system as they relate to low level debugging • How to display and set memory and process registers • Disassemble memory, and set execution breakpoints • Discipline, a tool for testing the validity of toolbox parameters • Debugging strategies you can use to find and cure common bugs • Includes MacsBug 6.2.

List $34.95 Our Price **$31.46** (BBUGREF)

Includes CD-ROM!

Check out our Web site!
• Full product descriptions
• Hundreds of more products
http://www.devdepot.com



Netscape Navigator 3.0

by Bryan Pfaffenberger

- Learn the best Web surfing techniques.
- Quickly find your way to valuable information on the Web.
- Learn to use the new plug-in tools, including Live3D and Real Audio.
- Send and receive electronic mail to anyone on the Internet.
- Utilize the enhanced security features for on-line shopping and ordering.

List $29.95 Our Price **$26.96** (BNETN3)

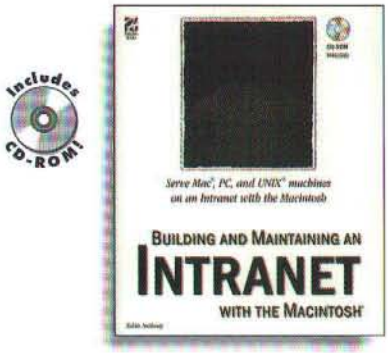
Programming For The Newton: Software Development using NewtonScript

by Julie McKeehan and Neil Rhodes. Foreword by Walter R. Smith

• An indispensable tool for Newton programmers. • Includes disk with sample Newton application from the books, as well as demonstration version of Newton Toolkit (NTK) – the complete development environment for the Newton®. • A Publication of AP Professional May 1994, Paperback, 393 pp.

List $29.95 Our Price **$26.95** (BPROGNEWT)

SEE RELATED CATEGORY: Dev. Environments



Building and Maintaining an Intranet with the Macintosh

by Tobin Anthony

- Set up services such as video conferencing, Real Audio, message boards, employee scheduling, e-mail, WWW, ftp, and many more.
- Find a simple, cost-effective solution to providing corporate information to company employees.
- Learn how to open portions of your intranet site up to the internet while keeping private information safe.
- Companion CD-ROM.

List $50 Our Price **$45.00** (BBAMAI)

BASIC for the Newton

by John Schettino & Liz O'Hara

- Program on Macintosh, Windows-based PC, or on the Newton itself.
- Straight-forward "programming by example" approach – you'll be writing Newton programs right away.
- Includes 3.5" disk containing Demonstration NS BASIC and over fifty example programs. (Newton not included)

List $35.95 Our Price **$32.35** (BNEWT)

SEE RELATED CATEGORY: Dev. Environments





Netscape Navigator Starter Kit for Macintosh by MacMillan

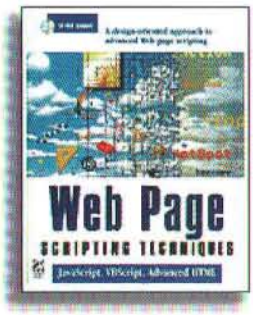
- Create your own home page in minutes and add hyperlinks to your favorite Web sites
- Add graphics, audio, and video
- Download files via FTP
- Load plug-ins to enable movies, audio, animations, and more

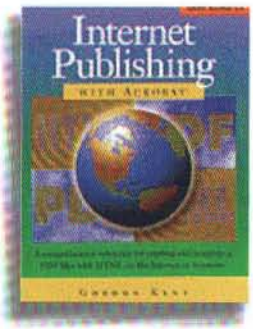
List $34.99 Our Price **$31.49** (BNETNSK)

Web Page Scripting Techniques by MacMillan

- Manipulating color effectively
- Placing images and using shapes
- Using JavaScript functions and objects, plus VBScript functions and objects
- Determining users' browser type and operating system

List $50 Our Price **$45** (BWEBPST)





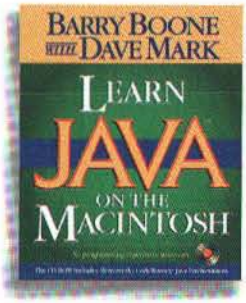
Internet Publishing with Adobe Acrobat

by MacMillan

- Optimizing PDF for Web viewing, integrating HTML and PDF, selecting PDF page dimensions, PDF file evaluation
- Setting links to PDFs and URLs, PDF forms, Security algorithms, Configuring Web servers for PDF, Embedded PDFs, PDF creation techniques.

List $40 Our Price **$36** (BIPWAA)

Order Toll-free
800-MACDEV-1
(800-622-3381)



Learn Java on the Mac

by Barry Boone with Dave Mark

- Easy-to-follow introduction for beginning programmers and Webmasters.
- Takes you through the core concepts of Java.
- Includes: Object-oriented techniques, unique features such as garbage collection, and basic programming concepts such as working with variables, threads and classes.
- Learn to apply this knowledge to write original Java applets for Web pages.
- Includes CD-ROM

List: $34.95 Our Price: **$31.45** (BLJAVA).

Providing Internet Services via the MacOS

Includes CD-ROM

by Carl Steadman and Jason Snell

- Shows you how to provide Web pages, FTP, Gopher, e-mail, and more with the Mac OS.
- Includes CD-ROM containing all the Mac server software and utilities you need.

List: $34.95 Our Price: **$31.46** (BPROVNET).

Learn HTML on the Mac

Includes CD-ROM

by Dave Mark and David Lawrence

- Shows you how to use HTML 3.0.
- Included CD-ROM contains Macintosh tools to design stunning multimedia Web pages.

List: $29.95 Our Price: **$26.95** (BLHTML)

WAIT... There's More!

Here are more products. For full product descriptions please see our Web site, or feel free to call, fax, or E-mail us. Our prices on books are at least 10% off list price.

PRODUCT	CODE	PRICE
evelopment Environments		
bleScript Applications: Building Applications w/FaceSpan	BAPSCAP	$31.45
sic for the Newton - Programming using NS BASIC	BNEWT	32.35
+ Programming with CodeWarrior	BCPPCW	32.35
+ Programming with MacApp	BCPPMACAP	31.46
C++ SDK User's Guide	BCPPUSER	29.00
deWarrior Inside PowerPlant	BINSPP	34.95
deWarrior Software Development using PowerPlant	BCWSWDEV	31.45
n Shafer Presents the Power of Prograph CPX	BDANPRO	19.95
RTRAN SDK User's Guide	BFORTUSER	34.95
ide CodeWarrior Book	BINSCW	34.95
st Resort Programmers Edition	BLSTRSRT	74.95
arn C on the Macintosh, 2nd Edition	BLEARNC2	31.45
stering the Think Class Library	BMASTERTCL	26.95
trowerks CodeWarrior Programming	BCWPROG	35.95
senting Magic Cap	BPRESMAGIC	15.25
ogramming for the Newton Using NewtonScript	BPROGNEWT	26.95
al World Apple Guide	BREALWLD	35.95
mantec C++ Programming	BSYMCPP	39.50
gent's Guide to Designing Programs	BTALIGENT	17.55
ual Programming with Prograph	BVISPRO	30.60
reless For The Newton	BWIRELESS	31.45

PRODUCT	CODE	PRICE
Hardware		
Apple CD-ROM Handbook	BCDHAND	14.36
Designing Cards & Drivers for the Macintosh	BCARD	26.96
LaserWriter Reference	BLASERREF	17.96
PCI System Architecture 3rd Edition	BPCISYS	31.46
PowerPC System Architecture	BPPCARCH	31.46
Internet Related		
1994 Internet White Pages	B94WHITE	26.95
Active Java	BACTJAVA	23.36
America Online for Dummies	BAOLDUM	17.95
CGI By Example	BCGIBE	31.45
Computer Privacy Handbook	BPRIV	22.45
E-Mail Essentials	BEMAILE	22.45
Elements of E-Mail Style	BEMAIL	13.45
Hooked on Java	BHJAVA	26.95
Instant Internet Guide	BINSTANT	13.45
Internet Book	BTHENET	22.50
Internet for Dummies 2nd Edition	BNETDUM2	17.99
Internet for Dummies Quick Reference	BDUMQCK	8.05
Internet for Macs for Dummies	BNETDUM	17.95
Internet for Macs for Dummies Bestseller Edition	BIFMFDBE	35.99

Internet Power Tools	BPWRTOOL	36.00
Internet Publishing with Adobe Acrobat	BIPWAA	36.00
Internet Secrets	BSECRET	35.99
Internet, The, Deluxe Edition	BNETDELUX	31.50
Intranet Web Dev.: Enterprise Alternatives to Client/Server	BINTWD	44.99
Java Essentials for C/C++ Programmers	BJAVAESSEN	17.95
Java in a Nutshell	BJAVANUT	13.45
Java Language API SuperBible	BJLAS	53.99
JavaScript for Macintosh	BJAVASCRPT	40.50
Learn HTML on the Macintosh	BLHTML	26.95
Learn Java on the Macintosh	BLJAVA	31.45
Mastering Netscape 4.0 for Macintosh, Second Edition	BMASNET4	36.00
More Internet for Dummies Starter Kit	BDUMNET	17.95
Mosaic for Dummies	BMOSDUM	17.99
Net Chat	BNETCHAT	17.00
NetObjects Fusion Handbook	BNETOFH	45.00
Netscape Navigator	BNETNAV	26.95
Netscape Navigator Starter Kit	BNETNSK	31.49
Perl Quick Reference	BPERLREF	17.99
Planning and Managing Web sites	BPLANWEB	35.96
Providing Internet Services	BPROVNET	31.46
Publish it on the Web	BWEBPUB	31.46
TCP/IP Vol 1-Vol 2 Bundle	BTCP12BNDL	99.00
Teach Yourself Java in 21 days	BJAVAMAC	36.00
Underground Guide to Telecommuting	BUNDER	22.45
Web Head Mac Guide	BWEBHEAD	22.45
Web Page Scripting Techniques	BWEBPST	45.00
Web Weaving	BWWEAV	22.45
Webmaster Macintosh	BWEBMAS	26.95

Scripting and Solutions

AppleScript Applications: Building Apps with FaceSpan	BAPSCAP	31.45
Applied Macintosh Scripting	BAPPLIED	31.45
Complete AppleScript Handbook	BAPLSCRHB	31.50
Complete HyperCard 2.2 Handbook	BHYPCRD2	31.50
Danny Goodman's Apple Guide Starter Kit	BDGAGSK	31.46
Danny Goodman's AppleScript Handbook	BDGASHB	35.00
HyperCard Stack Design	BHYPSTA	19.95
HyperTalk 2.2: The Book	BHYPTAL	31.50
JavaScript for Macintosh	BJAVASCRPT	40.50
Perl Quick Reference	BPERLREF	17.99
Real World Apple Guide	BREALWLD	35.95
Tao of AppleScript: BMUG's Guide to Macintosh Scripting	BTAO	26.95

Technical Reference

Active Java	BACTJAVA	23.36
Apple CD-ROM Handbook	BCDHAND	14.36
Art of Human Interface Design	BAHID	29.65
Black Art of Mac Game Programming	BBLACK	35.99
C++ for Dummies	BCPPDUM	17.95
C for Dummies Vol. 1	BCDUM	17.95
Developing Object Oriented Software for the Macintosh	BDEVOB	26.06
Essential OpenDoc	BESOD	35.95
Extending the Mac Toolbox	BETMT	22.46
Foundations of Macintosh Programming	BFOUND	35.96
Fragment of Your Imagination	BFRAG	35.96
Guide to Macintosh Software Localization	BLOCALIZ	24.26
Guide to Macintosh System 7.5	BSYS7.5	22.50
How to Write Macintosh Software	BWRITE	26.05
Inside AppleTalk	BAPTALK	31.45
Inside the Macintosh Communications Toolbox	BCOMM	22.45
LaserWriter Reference	BLASERREF	17.96
Learn C++ on the Macintosh	BLRNCPP	33.26
Learn C on the Macintosh, 1st Edition	BLEARNC1	31.45
Learn C on the Macintosh, 2nd Edition	BLEARNC2	31.45
Mac Programming for Dummies	BMACDUM	17.95
Macintosh C Programmer Primer Volume 1	BCPRIM1	24.25
Macintosh C Programmer Primer Volume 2	BCPRIM2	24.25
Macintosh OLE2 Prog. Reference Working with Objects	BOLE2	40.45
Macintosh Pascal Programming Primer Volume I	BPASCPRI	24.25
Macintosh Programming Secrets	BPSECRET	28.76
Macintosh Programming Techniques	BPTECH	31.95
More Mac Programming Techniques	BMORETECH	34.50
Network Frontiers Bundle	BNETFB	59.95
Newton Programming Guide	BNEWTPGUID	40.46
Object Oriented Programming Design	BOOPRODES	20.66
Optimizing PowerPC Code	BOPTPPC	35.96
PostScript Language Reference	BPSLANREF	29.66
Power Macintosh Programming Starter Kit	BPPCSTART	35.10
Powerbook: Digital Nomad's Guide	BPBTDNG	22.46
PowerPC Programmer's Toolkit	BPPCPT	40.50
Programming Introduction to the Macintosh Family	BFAMILY	22.46
Programming for System 7	BSYS7	24.25
Programming Primer Macintosh Volume 1	BPRIMMAC	34.15
Programming QuickDraw	BPROQDRAW	24.25
Programming Starter Kit	BPROSTART	40.50
Programming with AppleTalk	BPROAT	22.45
QuickTime - Official Guide for Macintosh Users	BQTGUIDE	45.00
QuickTime Starter Kit for the Macintosh	BQTSKIT	40.50
Real World Apple Guide	BREALWLD	35.95
ResEdit All Night Diner	BRESDINE	22.45
ResEdit Complete, 1st Edition	BRESED1	31.45
ResEdit Complete, 2nd Edition	BRESED2	31.45
ResEdit Reference	BRESEDREF	26.96
Software by Design: Creating User Friendly Software	BDESIGN	26.95
Symantec C++ for the Macintosh: The Basics	BSCFTMTB	31.46
Teach Yourself Macintosh C++ in 21 Days	BCPP21D	26.99
Technical Introduction to the Macintosh Family	BTITTMF	24.26
Tog on Software Design	BTOG	26.95
Wireless for the Newton Development for Mobil Comm.	BWIRELESS	31.45
Writing Localizable Software	BLOCAL	24.25

Miscellaneous

Adobe Premiere for the Macintosh	BPREM	44.95
America Online for Dummies	BAOLDUM	17.95
Art of Human Interface Design	BAHID	29.65
Best of MacTutor 3, 4, 5	BMTBEST	9.95
CD-ROM Guide to Multimedia Authoring	BCDMULTI	40.45
CompuServe for Dummies	BCSDUM	17.95
Creating Interactive CD-ROM	BINTERCDR	35.95
Cyberpunk Handbook	BCYBPUNK	8.95
Danny Goodman's Macintosh Handbook	BGOODHB	26.95
eWorld: The Essential Guide	BEWORLD	9.95
FrameWorks Source Code Disk	MTFWSC	9.95
FrameWorks Magazine Back Issue	MTFWBACK	8.00
Global Interface Design	BGLOBAL	32.35
Graphic Gems 2	BGEMS2	44.95
Graphic Gems 4	BGEMS4	44.95
Graphic Gems V	BGEMS5	44.95
Infini-D Revealed	BINFDREV	40.50
Late Night with MacHack	BLATE	26.95
Mac Bathroom Reader	BBATH	11.70
Mac Screamer: The Ultimate Macintosh Supercharging Kit	BSCREAM	31.50
Macintosh Crash Course	BCRASH	26.95
MacTech Back Issues	MTBACKISS	5.00
Macworld Ultimate Macintosh Book	BULTMAC	35.95
Managing AppleShare & Workgroup Servers	BMAWS	26.95
MADACON '93 CD-ROM	SMADA93	9.95
Multimedia Authoring: Building and Developing Documents	BMMAUTH	31.45
Multimedia Starter Kit for Macintosh	BMMSTART	27.00
Profit from Experience	BPROFIT	22.45
Sad Macs, Bombs and Disasters	BSADMAC	22.45
Sex, Lies & Video Games	BSEX	31.46
Stupid Mac Tricks	BSTUPIDMAC	17.95
The Software Developer's & Marketer's Legal Companion	BSDAMLC	33.26
Tricks of the Mac Game Gurus	BTRICKS	45.00
Zen and the Art of Resource Editing	BZAAORE	27.00

All entries in this index are alphabetized.
For your convenience the product names are **bold**, book names are *italicized* and company names are in plain text.

3D Graphics Programming Using QuickDraw 3D.....24
3D Graphics-Tips, Tricks, & Techniques.....24

A

A Fragment of Your Imagination.....21
Absoft.....6
Adianta.....12
Advanced Color Imaging on the Mac OS.....25
Apple Computer, Inc. 3, 4, 16, 28
Apple Dylan Technology Release.....4
Apple Media Tool Programming Environment 2.0...3
AppleGuide Complete.....25
AppleScript Applications: Building Applications with FaceSpan.....17
AppleScript Finder Guide, English Dialect.....22
AppleScript Language Guide.....22
AppleScript Scripting Additions Guide.....22
AppleScript Software Development Toolkit 1.1.....3
Applied Mac Scripting.....22
AppMaker.....14

B

B-Tree HELPER 2.2.....14
Bare Bones Software.....10, 13
BASIC for the Newton.....28
BBEdit 4.0.....10
Best of MacTutor.....2
Black Art of Macintosh Game Programming.....24
Bowers Development.....14
Building and Maintaining an Intranet with the Mac.....28

C

C++ Programming with CodeWarrior.....26
C++ Programming with MacApp.....26
CGI By Example.....19
CodeManager.....11
CodeWarrior 10 Gold.....5
CodeWarrior Software Development Using PowerPlant.....19
CodeWarrior Wear.....5
CPU Doubler.....11
CronManager.....11
Cyberdog Programmers Kit.....21

D

Dan Shafer Presents the Power of Prograph CPX.....27
Danny Goodman's AppleScript Handbook.....22
Danny Goodman's Apple Guide Starter Kit.....27
DeBabelizer: The Authorized Edition.....23
Designing Animation for the Web.....19
Discover Programming for Macintosh.....5

D

dtF.....13

E

Essential OpenDoc.....26

F

FaceSpan v2.1.....9
Fortner Research.....13
Fortran 77SDK.....6

G

Graphic Gems V.....25
Guarantee Terms.....1
Guide Composer 1.2.....11

H

Hooked on Java.....17

I

InCider.....13
Infini-D Revealed.....21
Inside CodeWarrior 10.....26
Inside Macintosh: AOCE Applications Interface.....16
Inside Macintosh: AOCE Service Module.....16
Inside Macintosh: CD-ROM.....16
Inside Macintosh: Devices.....16
Inside Macintosh: Files.....16
Inside Macintosh: Imaging with QuickDraw.....16
Inside Macintosh: Interapplication Communications.....16
Inside Macintosh: Macintosh Toolbox Essentials.....16
Inside Macintosh: Memory.....16
Inside Macintosh: More Macintosh Toolbox.....16
Inside Macintosh: Networking.....16
Inside Macintosh: Operating System Utilities.....16
Inside Macintosh: Overview.....16
Inside Macintosh: PowerPC Numerics.....16
Inside Macintosh: PowerPC System Software.....16
Inside Macintosh: Processes.....16
Inside Macintosh: QuickDraw GX Environ. & Util......16
Inside Macintosh: QuickDraw GX Graphics.....16
Inside Macintosh: QuickDraw GX Objects.....16
Inside Macintosh: QuickDraw GX Printing Extension....16
Inside Macintosh: QuickDraw GX Printing.....16
Inside Macintosh: QuickDraw GX Prog. Overview.....16
Inside Macintosh: QuickDraw GX Typography.....16
Inside Macintosh: QuickTime Components.....16
Inside Macintosh: QuickTime.....16
Inside Macintosh: Sound.....16
Inside Macintosh: Text.....16
Inside Macintosh: X-Reference.....16
Inside PowerPlant Manual.....25
Internet Publishing with Adobe Acrobat.....29
Intranet Web Development.....23

J

Java in a Nutshell.....17
Java Language API SuperBible.....21

JavaScript for the Macintosh 18

L

Late Night Software Ltd. 9
Learn C on The Macintosh Second Edition 19
Learn C++ on the Macintosh 20
Learn HTML on the Mac 29
Learn Java on the Mac 29

M

Mac OS 8 Revealed 18
Macintosh C Programming Primer Volume I 20
Macintosh C Programming Primer Volume II 20
Macintosh Pascal Programming Primer Volume I 20
Macintosh Programming Secrets 20
MacsBug Reference & Debugging Guide 27
MacTech Back Issues **2**
MacTech CD-ROM Volumes 1-11 ..... **2**
MacTech Magazine **2**
MacTech Mouse Pad **2**
Main Event Software 9
Mastering Netscape 2.0 for Macintosh 23
Mastering the THINK Class Library 27
Memory Mine **12**
Metrowerks CodeWarrior Programming 27
Metrowerks 5, 11
Movie Cleaner Pro 1.2 ..... **10**
Multimedia Authoring with Apple Media Tool **4**

N

Natural Intelligence 8
NeoAccess **11**
Neologic 11
NetObjects Fusion Handbook 21
Netscape Navigator 3.0 28
Netscape Navigator Starter Kit for Macintosh 28
Network Frontiers Bundle 23
Newton Programmer's Guide for Newton 2.0 3
Newton Toolkit 1.6 ..... **3**
Nisus Software 10
NS BASIC 3.5 for the Newton **7**

O

Object-Oriented Fundamentals 1.1 3
Object Oriented Program Design 23
Onyx Technology 12
OOFILE HTML Writer **8**
OpenDoc Programmer's Cookbook 25
OpenDoc Programmer's Guide 26
Optimizing PowerPC Code: Programming the PowerPC 19
Orchard Software 11

P

Paradigm Software 13
Perl Quick Reference 19
Personal MachTen for 68K **7**
PictureCDEF 1.3 ..... **13**
Planning and Managing Websites 18
Power MachTen–UNIX **6**
Power Macintosh Programming Starter Kit 19
PowerPC Programmer's Toolkit 23
PowerTap **13**
Presenting Magic Cap **7**
Programmer's Toolbox Assistant CD-ROM **17**
Programming For The Newton 28
Programming with AppleTalk 21
Providing Internet Services via the MacOS 29

Q

QC **12**
QUED/M 3.0 ..... **10**
QuickTime Developer's Kit 2.0 ..... **4**
QuickTime VR Authoring Tools Suite 2.0 ..... **4**

R

ResEdit All Night Diner 26
ResEdit Complete 25
Roaster Subscription **8**
Roaster **8**

S

Script Debugger **9**
Scripter 2.0 ..... **9**
ScriptGen Pro **14**
Seapine Software, Inc. 14
Sex, Lies and Video Games 24
SoftPolish **13**
Spellswell Plus 2.0.4 ..... **13**
Spyer **13**
Step-Up Installer Pack **14**
StepUp Software 11, 14
StoneTable Extra **12**
StoneTable **12**
Symantec C++ 8.5 ..... **6**
Symantec C++ for 68k **6**
Symantec Café **8**
Symantec Corporation 6

T

TCP/IP Scripting Addition **8**
Teach Yourself Java for Macintosh in 21 Days 18
Tenon Intersystems 6
Terms and Conditions 1
Terran Interactive 10
TestTrack **14**
The Elements of E-Mail Style 17
Think Pascal Version 4.0 ..... **6**
Tog on Software Design 23
Tricks of The Mac Game Programming Gurus 24

V

Visual Programming with Prograph CPX 27
VOODOO **12**

W

Web Page Scripting Techniques 28

X

Xplain Corporation 2

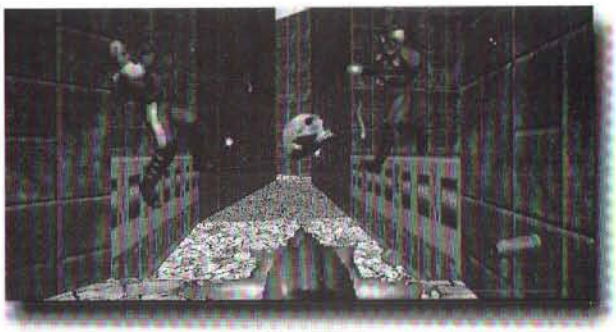
1-800-MACDEV-1 • Outside U.S. & Canada: 805-494-9797 • Fax: 805-494-9798

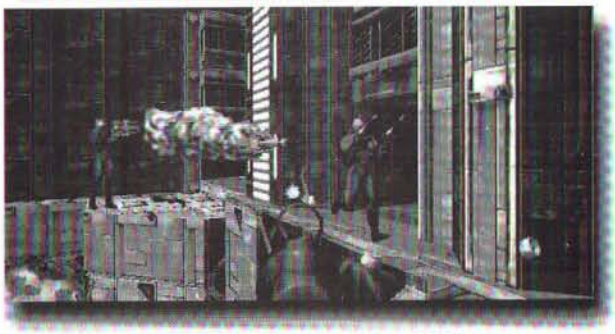
Behind every good Mac app...

Every job requires the proper tool. When it comes to crafting good code, **BBEdit 4.0** is the **Swiss Army Chainsaw** of text editors.

"If you have to spend time with text on the Mac, you should use BBedit."
-- MacWeek, September 30, 1996

- **Compare** source files and **apply differences**
- **Open From** and **Save To FTP**
- **Search** and **replace** multiple files using **Grep**
- Navigate source files with integrated **PopupFuncs™** technology
- **Browse** project documents





Bungie Software's **Marathon Infinity**, created using **BBEdit**.



Source code from **Marathon Infinity** displayed in **BBEdit**.

...is a great text editor.

BBEdit 4.0

MacUser ★★★★½

score card MacWEEK ◆◆◆◆◆

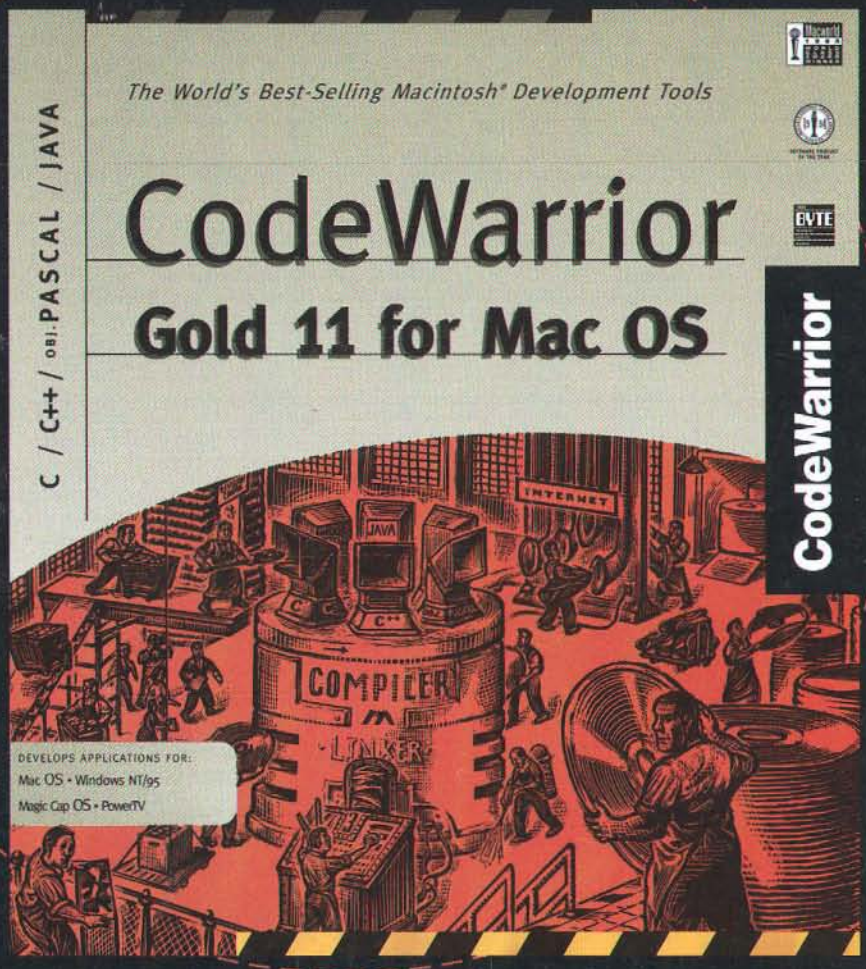
http://www.barebones.com/chainsaw

To order, call 617-778-3100

Bare Bones Software, Inc.
P.O. Box 1048, Bedford, MA 01730 • main 617-778-3100 • fax 617-778-3111

BBEdit is a trademark of Bare Bones Software, Inc. All other trademarks and registered trademarks are properties of their respective holders. © 1997 Bare Bones Software, Inc.

CodeWarrior 10 was so popular with developers that we're replacing it.



Blood, Sweat and Code

metrowerks

Three times per year, every year. We upgrade CodeWarrior Gold, the world's most popular and powerful Mac development tool suite, to keep up with the fast moving world of Mac developers. Now here's Version 11, our most powerful CodeWarrior yet. More reliable, more versatile, faster and more killer than ever before, but just as affordable. It's a result of your suggestions and our imagination.

CodeWarrior Gold 11 - $399

MW CodeManager 3 - Bring Order to Chaos.

Save time and money with the latest upgrade of the only cross-platform source code management system for the Mac. MW CodeManager 3 is based on and compatible with Microsoft VisualSourceSafe™ 4.0a. It works with source files as well as any type of binary file including graphic, database, library and executable files. With an improved interface and new Administrator application you can manage millions of lines of code in hundreds of files and maintain version control with ease. It's an essential tool for the well-equipped CodeWarrior.

MW CodeManager 3 - $399

For More Information Call 1-800-377-5416

System requirements: 68020 processor or higher, or PowerPC 601 processor or higher. 8 MB RAM minimum, 16 MB RAM recommended. CD-ROM drive required for installation.

All trademarks are the property of their respective owners and are hereby recognized. ©1996 Metrowerks Inc. All rights reserved.